



Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Afonso da Silva Batista

Railway Embedded Software Validation – Component Testing

Internship

Supervisors:

Professor Paulo Manuel Machado Coelho (IPT)

Engineer Pedro Miguel Marques Ferreira Serra (CSW)

Dissertation presented to Instituto Politécnico de Tomar to fulfill the requirements
necessary to obtain the Master's degree in Electrical Engineering

Instituto Politécnico de Tomar

June / 2021

I dedicate this work to those who never give up.

“Deus quer, o homem sonha, a obra nasce.”

Translation:

“God wills, the man dreams, and the work is born.”

Fernando Pessoa (Portuguese poet)

ABSTRACT

This report/dissertation is developed within the course of the Master's in Electrical Engineering, and for Internship Discipline, represents the work developed at Critical Software Company, in the context of Railway Embedded Software Validation in the field of Component Testing.

The project developed in this internship aims to test control system components of Train (Lights, Braking, ...), i.e., to test whether one of the parts of the components is functioning within the required parameters.

For this assignment it was necessary to go through a learning process in several stages, namely: - the way the trains work; - the way that standards are applied; - the way that necessary requirements are required for trains to work within safety parameters. With this in perspective, formal verification activities were carried out, with the objective of specifying and developing the various levels of testing the system.

Some of the systems that were studied focused on the thematic of traction, brake system, drive desk controls and diagnostics.

Keyword: Train, Railway, Component Testing, TCMS, Safety-critical

RESUMO

Este relatório/dissertação foi desenvolvido no âmbito do Curso de Mestrado em Engenharia Eletrotécnica, e para a Unidade Curricular de Estágio, e representa o trabalho desenvolvido na empresa Critical Software, no âmbito do projeto interno *Railway Embedded Software Validation* na área de *Component Testing*.

No projeto em que está envolvido este estágio, visa-se testar componentes do sistema de controlo do comboio (Luzes, Travagem, ...), ou seja, testar se uma das partes dos componentes está a funcionar dentro dos parâmetros exigidos e/ou estabelecidos.

Para isso foi necessário passar por um processo de aprendizagem com várias etapas, entre as quais se podem destacar: - como funcionam os comboios; - como são aplicadas as normas; - como são descritos os requisitos necessários para que os comboios funcionem dentro dos parâmetros de segurança. Com isso em perspetiva, foram realizadas atividades de verificação formal, com objetivo fazer a especificação e desenvolvimento dos diversos níveis de teste o sistema.

Alguns dos sistemas estudados foram de tração, sistema de travagem, controlos do motorista e de diagnóstico.

Palavras-Chave: Comboio, Ferrovia, Teste de Componentes, TCMS, Sistemas Críticos

ACKNOWLEDGMENT

I would like to begin giving my sincere thanks to Critical Software (CSW) that welcomed me at this internship, I am grateful for all the knowledge that they transmitted to me. I give my special thanks to the CSW Internship Supervisor, Engineer Pedro Serra, to all the support and availability that he provided me during this internship.

I also have a strong sense of appreciation for the team in which I was included, who supported me and gave me strength throughout this internship, with more focus in Ricardo Nunes (my tutor), who I am grateful for having supported me throughout the internship period and for all the wisdom he transmitted to me. I also would like to thank Mário Ferreira for the knowledge and support on Automatic Tool (Jenkins). Without forgetting to thank the Tomar Office of Critical Software for its welcome, despite the circumstances of the time of this internship.

I am grateful for the guidance and availability that my IPT Internship Supervisor, Professor Paulo Coelho, did have towards me, during my internship and in the preparation of this Report. I also would like to thank him for the support he gave me from the beginning, both at the internship level, and also before, at my academic journey.

A final thanks to Instituto Politécnico de Tomar (IPT), for the training that it provided me in this last years and for all the teachers that have passed on their knowledge to me until this moment. I also do not forget all the support I have received from family and friends during this period.

Table of Contents

ABSTRACT	VII
RESUMO.....	IX
ACKNOWLEDGMENT	XI
List of Figures	XVII
List of Tables.....	XXI
List of Abbreviations and Acronyms	XXIII
1. Introduction	1
1.1. Framework	1
1.2. Context.....	1
1.3. Objective	1
1.4. Planning	2
1.5. Report Structure	3
2. Brief Company History	5
2.1. CRITICAL Software.....	5
2.1.1. History	5
2.1.2. Culture.....	6
2.1.3. Railway	7
3. State of the Art and Standards.....	9
3.1. Railway	9
3.1.1. History	9
3.1.2. Rolling Stock Components.....	12
3.1.3. Other concepts of railway	24
3.2. Standards.....	26
3.2.1. EN50126/8/9	26
3.2.2. IEC 61508	29
3.2.3. DO178C	31
3.2.4. ISO 26262	32
3.2.5. ISO/IEC62443	33
3.2.6. ISO 27001	35
3.3. Testing techniques	36
3.3.1. Functional Testing	36

3.3.2. Black Box Testing	37
3.3.3. White Box Testing	39
3.3.4. Differences between Black box vs White Box Testing	41
3.3.5. Static program analysis	42
3.4. What is Component Testing?	43
3.5. What is Continuous Integration and Continuous Delivery (CI/CD)?.....	44
3.5.1. What is the difference between CI and CD?.....	44
3.5.2. Continuous Integration.....	44
3.5.3. Continuous Delivery.....	45
4. Workflow in CSW Project.....	47
4.1. Introduction	47
4.2. Analysis	48
4.2.1. Requirements analysis	49
4.3. Design.....	50
4.3.1. Test Case Design.....	51
4.3.2. Example TCD	53
4.4. Implementation.....	59
4.4.1. How Tests Are Performed	60
4.4.2. Example TS.....	60
4.4.3. Results	61
4.5. Execution and Report	62
5. Techniques used in the Project	63
5.1. Positive/Negative Tests	63
5.2. MC/DC	63
5.2.1. Used in process.	64
5.3. Boundary Value Analysis.....	65
5.3.1. Used in process.	66
5.4. Equivalence Class Partitioning.....	66
5.4.1. Used in process.	66
6. Problems and Solutions that appeared in the Project	69
6.1. What problems appear?	69
6.1.1. Real-time system simulation.....	70
6.1.2. Requirements with problems	71
6.1.3. Missing Signals in ISL.....	71

6.2.	Debug of Problems	71
6.2.1.	From TS.....	71
6.2.2.	CSW Client IDE	71
6.2.3.	Observe wave behavior.	72
7.	Automation Tool - Jenkins	73
7.1.	Jenkins	73
7.2.	Jenkins History	73
7.3.	Jenkins Pipeline	73
7.3.1.	Jenkins File.....	74
7.3.2.	Pipeline Syntax.....	75
7.3.3.	Advantages	76
7.3.4.	Jenkins Workflow.....	77
7.4.	Jenkins Architecture	79
7.4.1.	Jenkins Master.....	79
7.4.2.	Jenkins Slave.....	79
7.5.	Jenkins Multibranch Pipeline.....	80
7.5.1.	What is it?	80
7.6.	Jenkins in Project.....	81
7.6.1.	Why is Jenkins use in the project?.....	81
7.6.2.	Why use Multibranch.....	82
7.6.3.	Work Done	82
7.6.4.	Possible improvements	88
8.	Programs Used in the Project.....	91
8.1.	Languages used.....	91
8.1.1.	C#	91
8.1.2.	Groovy	92
8.1.3.	Python.....	93
8.2.	GIT.....	94
8.2.1.	What is Git?	94
8.2.2.	How Git works	95
8.2.3.	Glossary	95
8.3.	MS Build.....	96
8.4.	CSW client Tools.....	96
9.	Other Activities	97

9.1. Newcomers Training	97
9.2. “Internal Training”	97
9.3. Basic Training.....	97
9.3.1. Basic Training - Processes & Tools	98
9.3.2. Basic Training - Software Development Process.....	98
10. Conclusions	99
References	101
Appendix	107
• Appendix A.....	107
○ Basic Training - Processes & Tools Certificate.....	107
○ Basic Training - Software Development Process Certificate.....	108
• Appendix B.....	109
• Appendix C.....	113
○ First TCD.....	113
○ Second TCD.....	113
○ Third TCD.....	113
○ Jenkins File.....	113

List of Figures

Figure 1 - Company Logo. (Wikimedia Commons, 2020).....	5
Figure 2 - Clients of Critical Software. (CRITICAL Software_E, n.d.)	8
Figure 3 - Diolkos Wagonway – Greece. (Terminiello, n.d.).....	9
Figure 4 - Example Steam turbine locomotive. (Wikipedia_B, 2020).....	10
Figure 5 - Salamanca Locomotive created by Matthew Murray. (Wikipedia, 2018).....	10
Figure 6 - C&SLR locomotive. (Wikipedia_C, 2020)	11
Figure 7 - Petrol–electric Weitzer railmotor. (Wikipedia_D, 2020)	11
Figure 8 - Siemens & Halske train (Siemens, n.d.)	12
Figure 9 - Siemens & Halske train plan (Siemens, n.d.)	12
Figure 10 - Presentation photo. (Siemens, n.d.)	12
Figure 11 - Rolling Stock Components. (RailSystem, 2015)	13
Figure 12 - Aluminum Car Body (Aluminium Manufacture, n.d.)	13
Figure 13 - Car Body Fittings (The Railway Technical Website, n.d.).....	14
Figure 14 – Bogies (The Railway Technical_B, n.d.).....	14
Figure 15 - Power System (The Railway Technical_E, n.d.)	14
Figure 16 – Propulsion (The Railway Technical_C, n.d.).....	15
Figure 17 - Auxiliary Systems (Toshiba Railway India, n.d.).....	15
Figure 18 - Braking System (The Railway Technical_D, n.d.).....	16
Figure 19 - Interior of train (Bombardier_A, n.d.)	17
Figure 20 – TCMS (Leroy Automation, n.d.)	17
Figure 21 – PIS (Railway East, n.d.)	18
Figure 22 - Communication Systems (EV dynamic, n.d.).....	19
Figure 23 – Cabling (Railway-News, 2018).....	19
Figure 24 - Doors (Wongm's Rail Gallery, 2017)	19
Figure 25 - HVAC of Train. (Luger, Kallinovsky, & Rieberer, 2016)	20
Figure 26 - Tilt System (Hitachi Rail, n.d.).....	20
Figure 27 - Interior Lighting (LPA Group, n.d.)	21
Figure 28 – Coupler (Wikipedia_E, 2020)	21
Figure 29 - TCMS applied in train. (Bombardier_B).....	22
Figure 30 - Interconnection of different systems. (Bombardier_B)	22

Figure 31 - Balise. (RailSystem, n.d.)	24
Figure 32 - A schematic showing the basic architecture of a fixed block ATC system with its three main components. (The Railway Technical_A, n.d.)	25
Figure 33 - ATP System (Railway Signalling Concepts, 2019)	25
Figure 34 - Standards for Railway Application. (Wollny, 2017).....	26
Figure 35 - System Live Cycle in "V" Presentation (Wollny, 2017) (CENELEC, 2017) ..	27
Figure 36 - V- model for SW. (Turck, 2020)	28
Figure 37 - Model V for EN50129 (Li, 2019).....	29
Figure 38 - IEC 61508 derived documents. (AHMAD, 2020).....	30
Figure 39 - Resume of Table of SIL (Cross, n.d.).....	31
Figure 40 - Safety Level in DO178C. (Hilderman, 2014)	32
Figure 41 - Systems in Car. (Mannion, 2017).....	33
Figure 42 - IEC 62443 Series of Industrial Security Standard. (IEC, n.d.).....	35
Figure 43 - Process of ISO 27001. (XpertLync, n.d.)	35
Figure 44 - Example of Functional Testing. (Shah, n.d.).....	36
Figure 45 – Black-box Testing. (T, 2020).....	38
Figure 46 - Black box testing. (QATestLab, 2018).....	39
Figure 47 - White Box Testing. (Dobra, 2019)	39
Figure 48 - Flow Static Analysis. (Mezquita, 2020)	43
Figure 49 - Representation of CI/CD (Red Hat, n.d.).	44
Figure 50 - Continuous Integration. (PagerDuty, n.d.)	45
Figure 51 - Continuous Delivery. (ATC Team, 2019).....	45
Figure 52 - Workflow Steps.	47
Figure 53 - Workflow in Project Summary.....	48
Figure 54 - Workflow in analysis phase.....	49
Figure 55 - Design Step.....	51
Figure 56 - TCD Components.....	51
Figure 57 - Example of Test Case Design.....	52
Figure 58 - Implementation step.	59
Figure 59 - Schematic in example of BVA. (Sharma, 2019)	65
Figure 60 - Observe waves.....	72
Figure 61 - Example of Jenkins file. (Jenkins_C)	74

Figure 62 - Example of Blue Ocean (Jenkins_B, n.d.).....	76
Figure 63 - Pipeline Flow. (Jenkins_C).....	77
Figure 64 - Flow diagram of Continuous Integration with Jenkins. (Saurabh, 2020).....	77
Figure 65 - Jenkins Architecture. (JavaTpoint, n.d.).....	79
Figure 66 - Jenkins internal Workflow. (JavaTpoint, n.d.)	79
Figure 67 - Multibranch Pipeline example. (Jenkins_D, n.d.)	80
Figure 68 - Process of TS.	83
Figure 69 - Multibranch process.....	85
Figure 70 - C# Logo. (Techbaz, n.d.)	91
Figure 71 - Groovy Logo. (Wikipedia_F, 2021)	92
Figure 72 - List of resumes of vantages. (Apache Groovy, n.d.)	92
Figure 73 - Python Logo. (Python, n.d.).....	93

List of Tables

Table 1 - Rolling Stock Components.	13
Table 2 - Safety Integrity Level (Borges, 2017).....	30
Table 3 - Differences between Black Box Testing vs White Box Testing. (Jain, 2020).....	41
Table 4 - Table of Resume of Component Testing.	43
Table 5 - Example of Requirement.....	49
Table 6 - Table Glossary of Requirement.	50
Table 7 - Requirement 1.	53
Table 8 - Requirement 2.	53
Table 9 - Table of Actions Realized for Requirement present.	54
Table 10 - Examples of code realized.	60
Table 11 - Results Table of results.	61
Table 12 - Positive/Negative tests	63
Table 13 - Example of MC/DC.	64
Table 14 - Example of BVA.....	65
Table 15 - Equivalence Partitioning	66
Table 16 - Principal problems in Process.	69
Table 17 - Non-use and Use of Jenkins. (Saurabh, 2020)	78
Table 18 - Advantages and disadvantages Jenkins in the project.....	81
Table 19 - Glossary of GIT.....	95

List of Abbreviations and Acronyms

AI - Artificial Intelligence

ATC - Automatic Train Control

ATO - Automatic Train Operation

ATP - Automatic Train Protection

ATS - Automatic Train Supervision

BCU - Brake Control Unit

BVA - Boundary Value Analysis

CBTC - Communications-Based Train Control

CCU - Computer Control Unit

CD - Continuous Delivery

CI - Continuous Integration

CR – Change Request

CENELEC - European Committee for Electrotechnical Standardization

CSW - Critical Software

DAL - Design Assurance Level

DMI - Driver-Machine Interface

EA - Enterprise Architecture

ECN - Ethernet Consist Network

ECP - Equivalence Class Partitioning

EN - European Standards

ERTMS - European Railway Traffic Management System

ETB - Ethernet Train Buses

ETCS - Electronic Train Control System

FR - Foundational Requirements

FR - Feature Request

HMI - Human Machine Interface

HVAC - Heating, Ventilating and Air Conditioning

IACS - Industrial Automation and Control Systems

IEC - International Electrotechnical Commission

I/O - Input/Output

IDE - Integrated Development Environment
IP - Internet Protocol
IPT - *Instituto Politécnico de Tomar*
ISA - Independent Safety Assessment
ISL - Interface Signal List
ISMS - Information Security Management
ISO - International Organization for Standardization
JVM - Java Virtual Machine
MC/DC - Modified Condition / Decision Coverage
MCG - Mobile Communication Gateway
MIO - Modular Input/Output
MVB - Multifunctional Vehicle Bus
OEM - Original Equipment Manufacturers
PIS - Passenger Information System
PR – Pull Request
TCMS - Train Control Management System
TC - Test Case
TCD - Test Case Design
TCU - Traction Control Unit
TS - Test Script
TSI - Technical Specifications for Interoperability
RAMS - Reliability, Availability, Maintainability and Safety
SIL - Safety Integrity Level
SDP - Software Development Process
SR - System Requirements
STU - System Under Test
SW - Software
UxD - User Experience Design
V&V - Verification and Validation
WTB - Wired Train Bus
WSP - Wheel Slip Protection

1. Introduction

1.1. Framework

The Internship is incorporated in the railway industry, where Critical Software provides safer, more flexible, and faster rail networks that support the worldwide needs for the evolution of rail transport.

1.2. Context

The railway industry is a highly competitive industry, where innovation rules and the Original Equipment Manufacturers (OEM) constantly improve safety, automation, and interoperability standards. Critical Software's vision provides solutions that meets the requirements of the Rail Industry, aiming safer, more flexible, and faster rail networks.

Critical Software supports manufacturers of railway material (circulating and non-circulating) in the development, testing and certification of embedded systems according to CENELEC EN50126/8/9 standards up to the highest level of safety integrity, SIL4.

Due to the knowledge in the field of track equipment and rolling stock control systems, CSW (Critical Software) have a substantial experience in working with CENELEC standards. CSW also have specialized knowledge of European Railway Traffic Management System (ERTMS), and Communications-Based Train Control (CBTC) and other technologies derived from it.

In the area of systems integration, "turnkey" maintenance solutions are presented based on the condition that complements the embedded system resources. This supports operators to add value to the end customer experience and allows a more efficient use of each operator's rolling stock fleet.

1.3. Objective

The objective of this internship is the development and validation of software (SW) directed to the railway area. In these tasks/project, I will approach the different phases of the SW development life cycle, as part of a team, to carry out the proposed work and integrate it in the company.

Thus, techniques and methodologies will be used in the validation of the SW components, developed by CSW, and in the development of documents with the test cases and the respective scripts. Subsequently, these scripts and test results must be integrated into the automation tool to allow the automatic execution of the tests.

1.4. Planning

The internship involves the following activities and respective tasks:

- T1 - Analysis of the technologies being developed and the international standards applicable to the development of embedded systems for the railway industry.
 - Internal training:
 - TCMS (Train Control Management System) architecture and its complexity.
 - Development environment (Programs in use in projects).
 - Development standards safety-critical:
 - IEC61508
 - EN50126/8/9
 - DO178C
 - ISO26262
 - Security for industrial automation and control systems:
 - ISO/IEC62443
 - Information security management:
 - ISO27001
- T2 - Ramp-up in project
 - Introduction to the project, the techniques and tools used in the tests:
 - Testing techniques
 - Functional testing
 - Black box testing
 - Static software analysis
 - Test methodologies
 - Automation server
- T3 - SW validation within the project

- Application of the techniques and methodologies previously studied in the SW of the train:
 - Analysis of software requirements
 - Specification of test cases
 - Programming of scripts to perform component tests
 - Execution of the tests and evaluation of the results
- T4 - Automatic execution of tests
 - Integration of test scripts in the automation tool

1.5.Report Structure

This report is divided into 9 chapters and 3 appendices. The organization of the report presents a sequential way of approaching the several subjects. The first chapter, “Introduction”, consists of the framework and context, objectives, planning, and the structure of the report. In chapter 2, “Brief Company History”, there is company introduction, considering history, culture, and values. In Chapter 3, “State of the Art and Standards”, we will cover the importance of the railway in the world society and some of the standards used in railway. The chapter 4, “Workflow in CSW Project”, focuses on the normal work of the project. The chapter 5, “Techniques used in the Project”, introduce the principal test techniques use in project. The chapter 6, “Problems and Solutions that appeared in the Project”, reference some problems that appeared during process, and resolutions. The chapter 7, “Automation Tool - Jenkins”, presents Jenkins and all necessary tools to work. The chapter 8, “Programs Used in the Project”, present some programs, tools and programming languages used or investigated. In Chapter 9, “Other Activities”, are presented the training programs attended during an internship in CSW. Finally, in Chapter 10 the Conclusions will be presented, and the most important aspects of the Internship will be highlighted.

This report was structured to present all the results of the work developed, and all the achievements gained from the knowledge acquired.

2. Brief Company History

In this chapter we will briefly cover the company's history.

2.1.CRITICAL Software

2.1.1. History

Critical Software's history begins in Coimbra University, where three engineers met while completing their PhDs in Computer Engineering. Together, Diamantino Costa, Gonçalo Quadros and João Carreira wrote several technical articles that attracted the attention of Jet Propulsion Lab (NASA). And thus, in 1998, Critical Software was born (CRITICAL Software_A, n.d.).

As a result, they have grown and now they provide systems and software services for safety, mission, and business-critical applications in several markets, including aerospace, defense, automotive, railway, telecoms, finance, and energy & utilities. Core competencies include system planning and analysis, system design and development, embedded and real-time systems, command & control systems, security and infrastructure, systems integration, business intelligence, independent software verification & validation, User experience design (UxD), Artificial intelligence (AI), digital transformation and smart meter testing. (CRITICAL Software_B, n.d.) (Wikipedia_A, 2020)



Figure 1 - Company Logo. (Wikipedia Commons, 2020)

In 2019 the Company had more than 900 employees and the revenues reached 58 million euros. (Wikipedia_A, 2020). Currently according to data given by the company there are more than 950 employees. (Critical Software_C, n.d.)

2.1.2. Culture

Critical Software revises itself in these values, betting on a tripartite strategy based on three foundations (Sousa, 2010):

1. Competence and dedication to processes and projects, to win potential future customers and build a base of recognition and merit.
2. Development of processes that allow creating a business commitment between the areas of production, qualification, and innovation, with gains in knowledge for the company and cost reduction for most customers.
3. Internationalization of the company, with opening of offices and points of interest in places where, on the one hand, they can boost and probe new customers and, on the other hand, create new development centers and potential centers of creativity and innovation.

Quality has a central strategic importance since the formation of the company. Has the company's Quality Director said: “We commit ourselves to producing the very best software on Earth and beyond, delivering real business value to our customers and partners.”.

The goals to be achieved are the following (CRITICAL Software_D, n.d.):

1. Establishing the right working mindset and use appropriate processes, tools, and technologies **“to ensure that we always achieve and exceed the highest standards”**.
2. Creating high-performance software that is **“secure, reliable and is a pleasure to use”**.
3. Responding quickly and efficiently - with agility - meeting customer demands and **“guaranteeing the highest quality outcomes”**.
4. Focusing on adding value to our customers – **“satisfaction is not just hoped for, it’s mandated”**.
5. Search continually for ways to **“improve our performance and how we work”**.
6. Align with **“high ethical standards and conform to all industry regulations”** –not taking shortcuts and looking beyond requirements.
7. **“Care about our people and help them to grow their skills”** so that they can enjoy what they do – which means they are engaged and want to do their best.

In short, CRITICAL Software assumes quality as its main objectives, to constantly improve.

2.1.3. Railway

In Railway industry, CSW does the following jobs (CRITICAL Software_E, n.d.):

- ✓ **Embedded Software Development** (CRITICAL Software_H, n.d.)
 - Production of requirements.
 - Architecture, design, and coding.
 - Verification and Validation activities.
- ✓ **Safety-Critical V&V** (CRITICAL Software_F, n.d.):
 - TCMS Software Testing: including component and integration testing in full conformance with the provisions of CENELEC EN 50128.
 - HMI Testing: testing of human machine interfaces (HMI) such as the software applications running on the consoles operated by the train driver.
 - System Testing: definition of test cases for different vehicle functions and execution of test procedures in either a test rack or train simulator. Verification of functions including main line voltage, auxiliary power, doors control, TCMS, etc.
 - In-Vehicle Testing: definition of test cases for tests on the vehicle and execution of the respective test procedures, which is done in close co-operation with the client's technical and management personnel.
- ✓ **RAMS & Certification Support** (CRITICAL Software_G, n.d.)
 - Project gap analysis: detection of gaps with respect to the safety requirements, namely those in EN 50126, and determination of the best way to overcome them, given specific project and client needs.
 - Safety management & safety case preparation: management of functional safety across the lifecycle, for both rolling stock and signalling systems, including the elaboration and management of the safety case.
 - Independent Safety Assessment (ISA): compilation and review of all project safety artefacts in conformation with the applicable regulation (both international standards and national regulation).
 - Training on functional safety: addressing the provisions of EN 50126, EN 50128 and EN 50129, either focusing on global normative aspects, project-specific needs, or both.

Introducing some of CSW's customers, at Railway level (CRITICAL Software_E, n.d.):

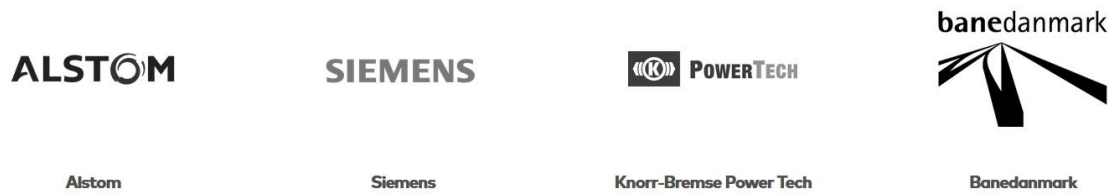


Figure 2 - Clients of Critical Software. (*CRITICAL Software_E, n.d.*)

3. State of the Art and Standards

This chapter will cover the importance of the railway in the world society and some of the standards used in railway and mentioned in the internship plan.

3.1. Railway

3.1.1. History

Railways are very important for today's transportation, of people and products, and society can't be imagined without it. Railways are transportation devices that have revolutionized our industry, human expansion, and the way we can move from place to place. (Train History, n.d.)

As referred in (Train History, n.d.), the first railway started to be used 2000 years ago, in ancient civilizations of Egypt, Babylon, and Greece. It intended to transport people and goods, that at that time was done with carts pulled by animals (horses or bulls). Quickly they noticed that animals spent much less energy if the cart was traveling on predetermined path, without possibility for steering or traveling over uneven terrain. The first railway was thus created.



Figure 3 - Diolkos Wagonway – Greece. (Terminiello, n.d.)

The figure above is the Diolkos Wagonway, an example of these ancient stone etched “wagonways” that can be found in the Isthmus of Corinth, Greece.

In the 18th century, each mine in Britain had its own simple rail network, with horses pulling wagons from mines to factories. The changes in this type of transport occurred in 1777, after the world found the incredible discovery, that was stationary steam engine, by James Watt's. (Train History, n.d.).

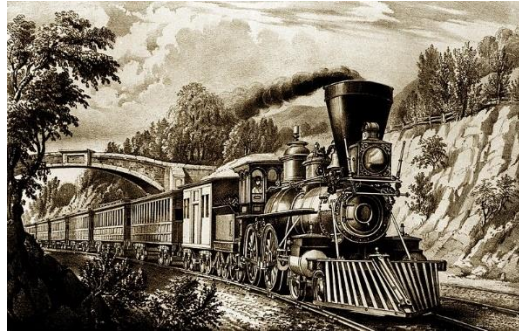


Figure 4 - Example Steam turbine locomotive. (Wikipedia_B, 2020)

First steam engines started running along primitive rail tracks in 1804, by Matthew Murray which was the first person to showcase a simple locomotive. With these locomotives the exploration of the railway all over the world began.

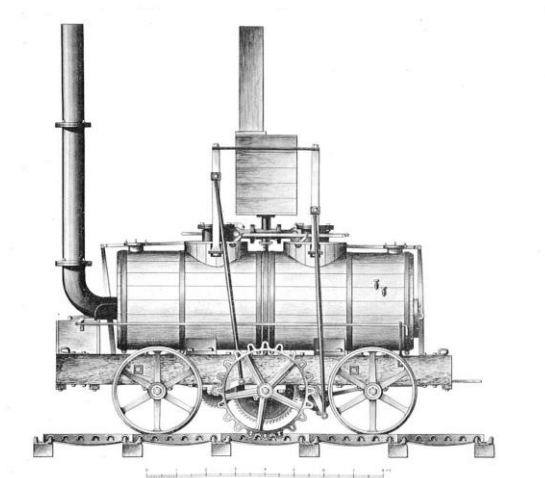


Figure 5 - Salamanca Locomotive created by Matthew Murray. (Wikipedia, 2018)

As train technology received massive updates over those first few decades of public work, urban engineers in London started formulating the first plans for inter-city railway tracks and underground tunnels. In 1890 the entire London train fleet started to use electrical engines. This was the beginning of electric locomotive.



Figure 6 - C&SLR locomotive. (*Wikipedia_C*, 2020)

Another important step in the history of trains was the introduction of Diesel engines, which brought to the end of steam locomotives age. After Second World War, due to be faster, easier to maintain and reliable, diesel trains started to be used in most of the world.

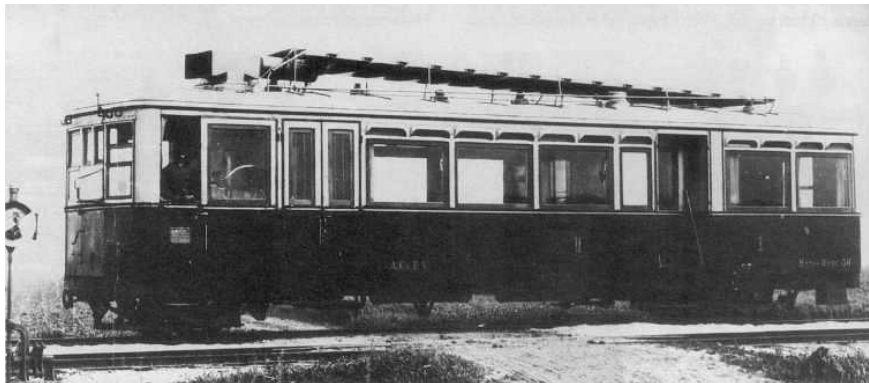


Figure 7 - Petrol–electric Weitzer railmotor. (*Wikipedia_D*, 2020)

The steam locomotive, as well as diesel locomotive, were not a solution for the use in cities since they proved to be too inflexible and to increase the pollution and noise in urban environment, so they started to be used only for long-distance travel. (Train History, n.d.).

On May 31, 1879, Siemens & Halske presented the world's first electric train in which power was supplied by a catenary. The 150-volt direct current flowed through the two rails to the small locomotive via an insulated flat iron bar mounted between the rails. (Siemens, n.d.)

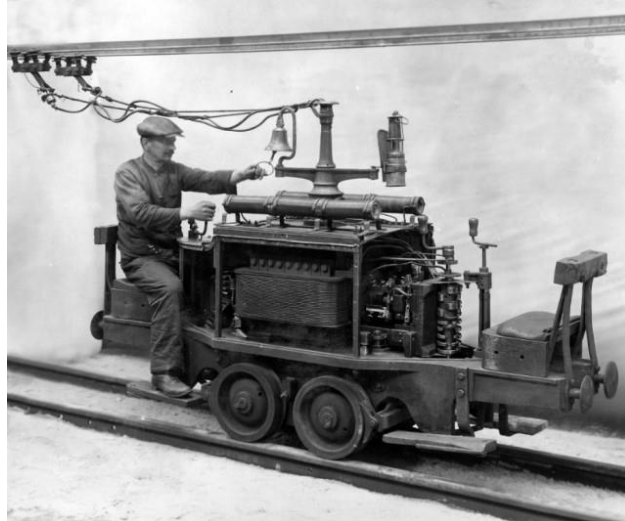


Figure 8 - Siemens & Halske train (Siemens, n.d.)

The little electric locomotive (Figure 8), on which the driver sat, could carry three carriages, each holding six passengers.

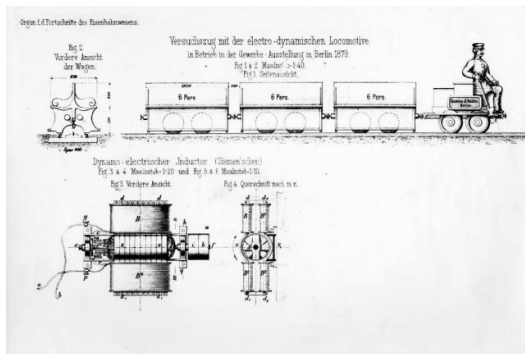


Figure 9 - Siemens & Halske train plan (Siemens, n.d.)



Figure 10 - Presentation photo. (Siemens, n.d.)

Today, trains represent one of the most important forms of transporting people and goods. Capital cities and big cities cannot live without underground metro systems, one of the main transportations within cities. Trains carry over 40% of worldwide goods between towns, countries, and continents. (Train History, n.d.)

3.1.2. Rolling Stock Components

Rolling stock refers to the vehicles that move on a railway. It generally includes motorized and non-motorized vehicles, for example for locomotives, railroad cars, coaches, and wagons. (RailSystem, 2015)

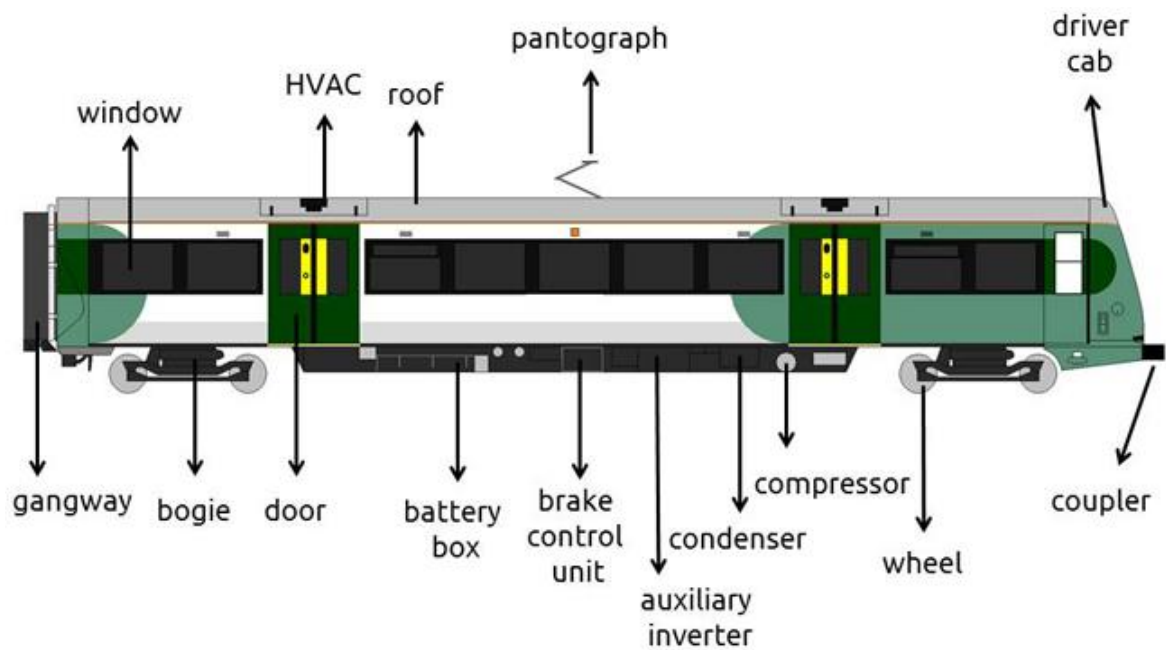


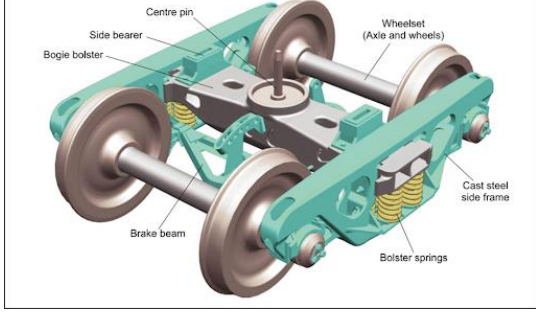
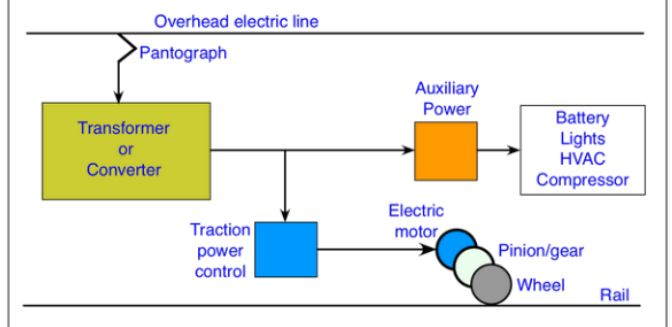


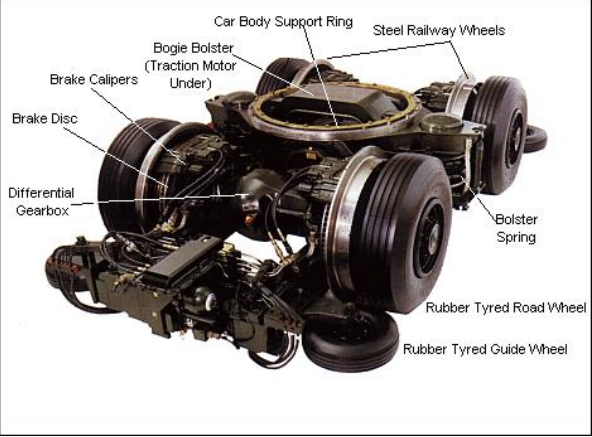
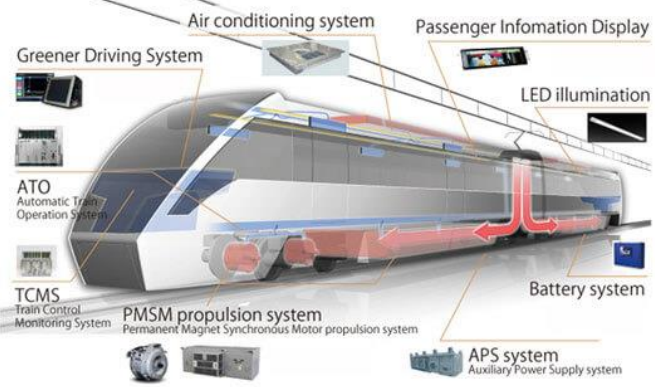
Figure 11 - Rolling Stock Components. (RailSystem, 2015)

Description of Rolling Stock Components (RailSystem, 2015):


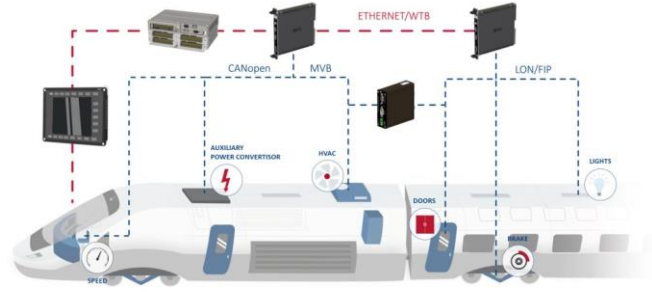
Table 1 - Rolling Stock Components.

Area	Component	
Car body	<ul style="list-style-type: none">• Car body shell• Underframe• Floor• Crash structure• Windows• Insulation• Painting and Sound damping• Fixing / connecting elements	<div><p>Figure 12 - Aluminum Car Body (Aluminium Manufacture, n.d.)</p></div>

Car body fittings	<ul style="list-style-type: none"> • Front/End car body fittings • Roof car body fittings • Underframe car body fittings • Lateral car body fittings 	 <p>Figure 13 - Car Body Fittings (<i>The Railway Technical Website, n.d.</i>)</p>
Guidance (Bogies and running gear)	<ul style="list-style-type: none"> • Motor bogie • Trailer bogie 	 <p>Figure 14 – Bogies (<i>The Railway Technical_B, n.d.</i>)</p>
Power System	<ul style="list-style-type: none"> • Power supply • Power generation • Power conversion • Power dissipation • Power storage 	 <p>Figure 15 - Power System (<i>The Railway Technical_E, n.d.</i>)</p>


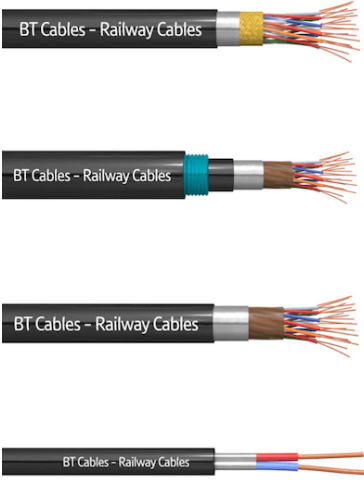

Propulsion	<ul style="list-style-type: none"> • Traction Control Unit (TCU) • Gear box • Traction motor • Mechanical transmission • Power converter 	 <p>Figure 16 – Propulsion (<i>The Railway Technical_C, n.d.</i>)</p>
Auxiliary systems	<ul style="list-style-type: none"> • Air supply system • Hydraulic system • Auxiliary electric system • Main Auxiliary Converter equipment • Low Voltage Power Supply / Battery Charger equipment • Special Aux. Converter equipment • Battery equipment • External supply system 	 <p>Figure 17 - Auxiliary Systems (<i>Toshiba Railway India, n.d.</i>)</p>

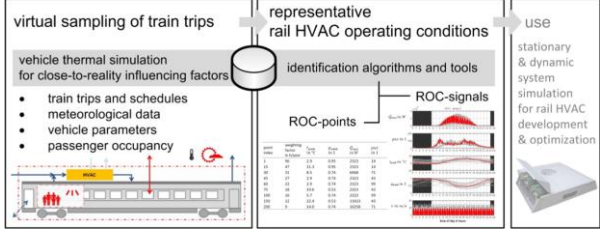
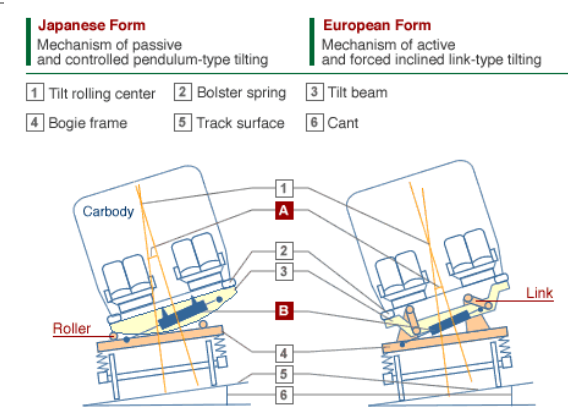
	<ul style="list-style-type: none"> • Cooling unit for power and drive systems • Fire protection system • Sanding equipment • Horn • Flange lubrication device 	
Braking System	<ul style="list-style-type: none"> • Brake control Unit (BCU) • Friction brake equipment • Wheel Slip Protection (WSP) equipment • Magnetic track brake equipment • Emergency brake equipment • Eddy current brake equipment 	<p>Figure 18 - Braking System (<i>The Railway Technical_D, n.d.</i>)</p>



Interiors	<ul style="list-style-type: none"> • Interior architecture • Interior equipment • Toilet system • Engine room • Catering system 	 <p>Figure 19 - Interior of train (<i>Bombardier_A, n.d.</i>)</p>
On board vehicle control	<ul style="list-style-type: none"> • Electronic Train Control System (ETCS) • Train Control Management System (TCMS) • Heritage Automatic Train Protection (ATP) unit • Automatic Train Operation (ATO) unit • Fault data logger • Heritage juridical recorder unit • Voice recorder 	 <p>Figure 20 – TCMS (<i>Leroy Automation, n.d.</i>)</p>

	<ul style="list-style-type: none"> • System, capture unit • Video surveillance • Electronic rear mirror 	
Passenger information System (PIS)	<ul style="list-style-type: none"> • Public Address System • Safety Alarm Systems • Central Passenger information System (PIS) unit • Driver-Machine Interface (DMI) for train/travel information • Seat reservation • Billing System • Trackside equipment 	 <p>The image shows the logo for Suzhou East Railway Co., Ltd. at the top, featuring the company name in English and Chinese, along with certification logos for IAF and CNAS. Below the logo is a 3D diagram of a train car with various PIS components labeled, including a central unit, DMI, and trackside equipment. The website address www.eastrailway.com is also visible.</p>

Figure 21 – PIS (Railway East, n.d.)

Communication systems	<ul style="list-style-type: none"> • Train to wayside communication system • Onboard communication system 	 <p>Figure 22 - Communication Systems (<i>EV dynamic, n.d.</i>)</p>
Cabling and Cabinets	<ul style="list-style-type: none"> • Cabling • Cabinets 	 <p>Figure 23 – Cabling (<i>Railway-News, 2018</i>)</p>
Door System	<ul style="list-style-type: none"> • External doors • Internal doors 	 <p>Figure 24 - Doors (<i>Wongm's Rail Gallery, 2017</i>)</p>

<p>Heating, Ventilating and Air Conditioning (HVAC)</p>	<ul style="list-style-type: none"> • Heating, Ventilating and Air Conditioning control unit (HVAC) • Air conditioning system • Heating system • Air ventilation and distribution system • Air intake • Exhaust air unit • Pressure protection system 	 <p>virtual sampling of train trips</p> <p>vehicle thermal simulation for close-to-reality influencing factors</p> <ul style="list-style-type: none"> • train trips and schedules • meteorological data • vehicle parameters • passenger occupancy <p>representative rail HVAC operating conditions</p> <p>identification algorithms and tools</p> <p>ROC-points</p> <p>ROC-signals</p> <p>use stationary & dynamic system simulation for rail HVAC development & optimization</p> <p>Figure 25 - HVAC of Train. (Luger, Kallinovsky, & Rieberer, 2016)</p>
<p>Tilt System</p>	<ul style="list-style-type: none"> • Tilt Control Unit • Actuating System • Pantograph tilt system • Tilt monitoring and detection 	 <p>Japanese Form Mechanism of passive and controlled pendulum-type tilting</p> <p>European Form Mechanism of active and forced inclined link-type tilting</p> <p>1 Tilt rolling center 2 Bolster spring 3 Tilt beam 4 Bogie frame 5 Track surface 6 Cant</p> <p>Carbody Roller Link</p> <p>Figure 26 - Tilt System (Hitachi Rail, n.d.)</p>

Lighting	<ul style="list-style-type: none"> • Emergency lighting system • Exterior lighting system • Interior lighting system 	 <p>Figure 27 - Interior Lighting (<i>LPA Group, n.d.</i>)</p>
Coupler	<ul style="list-style-type: none"> • Front coupler • Intermediate coupler • Emergency coupler (Towing coupler) 	 <p>Figure 28 – Coupler (<i>Wikipedia_E, 2020</i>)</p>

Most of the components need to be controlled by the train systems, therefore they need to be tested in software.

3.1.2.1. TCMS

Train Control & Management System (TCMS) is a train distributed control system. It has the capability to control all the systems on board, connecting all house systems in a train in a secure and fault-resistant manner. (Tew, 2015)

This system contains the following functions (Bombardier_B):

- ✓ Train control and command.
- ✓ Train safety.
- ✓ Maintenance.
- ✓ Passenger Information Systems.
- ✓ Passenger Comfort system.
- ✓ Video surveillance.
- ✓ Train to wayside data transfer.

This system (TCMS) aims to replace complicated systems (various equipment and wiring), by a simpler system.

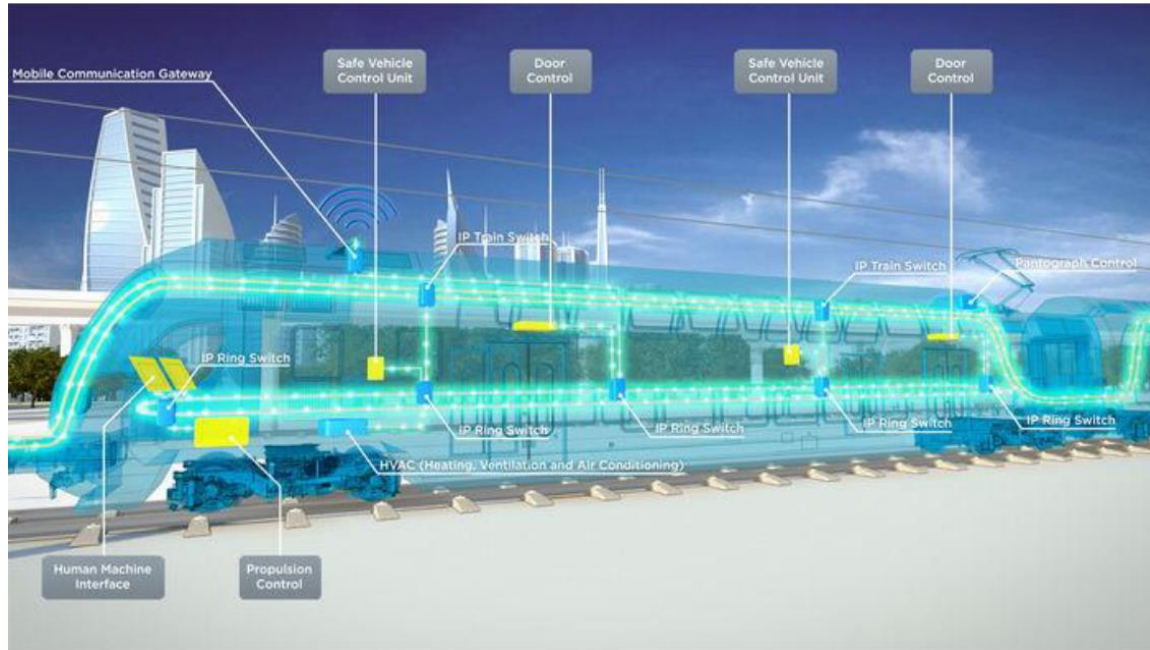


Figure 29 - TCMS applied in train. (Bombardier_B)

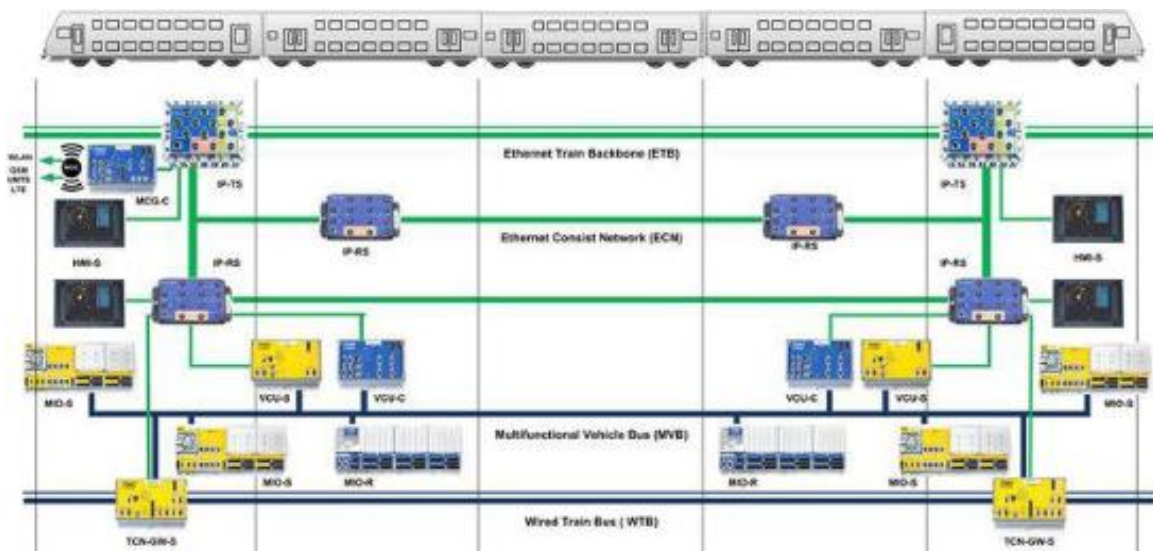


Figure 30 - Interconnection of different systems. (Bombardier_B)

- TCMS Components (Tew, 2015):
 - **Computer Control Units** (CCUs) are programmable devices, they run software in real-time, which allows for predictable operational performance and control.
 - **Human Machine Interfaces** (HMIs) provide a high-resolution and touch-screen programmable interface to relay information to drivers and maintainers. They can be customized to suit developing operational

circumstances and can provide redundancy for other screen-based functionality that may be required in the driver's cab.

- **Modular Input/Output** (MIO) devices provide a flexible combination of digital and analogue I/O, allowing to future expansion and facilitating a wide variety of control and monitoring solutions.
- **Mobile Communication Gateways** (MCGs) typically provide Communication for train, combined GSM-P, Wi-Fi, and GPS capabilities, making the train location available to both onboard and wayside services.

Safety-certified variants of CCUs and MIOs can be used to deliver safety-critical functionality. In Figure 30 is represented in yellow the Safety equipment (CCU-S for example), and in blue the non-Safety equipment (CCU-O for example).

The devices described above will be connected to one or more networks. Figure 30, presents a diagram that resumes an example of networks inter systems. Other field-bus technologies exist but are generally adopted by other industries such as automotive or building-control and they are not the subject of standardization for rail. (Tew, 2015)

- TCMS Networks (Tew, 2015):
 - The **Multifunctional Vehicle Bus** (MVB) provides device connectivity within a consist, i.e., a set of vehicles which can be operated as a train independently. Depending on the number of devices in the consist, there may be multiple buses. MVB is a managed serial protocol, often implemented on the RS485 physical layer.
 - An **Ethernet Consist Network** (ECN) is typically organized in a ring topology to provide redundant paths in the event of cable or switch failure. It provides an Internet Protocol (IP) interface to TCMS and other systems within a consist.
 - A **Wired Train Bus** (WTB) provides redundant connectivity between coupled consists, based on an RS485 physical layer, while Ethernet Train Buses (ETB), which are typically organized as redundant linear buses, provide IP connectivity between coupled consists.

3.1.3. Other concepts of railway

- **Balise** - Balise is an electronic beacon or transponder placed between the rails of a railway as part of an automatic train protection system (ATP). Transmission device (passive transponder) that can send messages to an on-Board subsystem. The on-board system tracks the train's location by counting wheel rotations (encoder) and correcting local of train in line (because of the slipping of the wheels). (RailSystem, n.d.)



Figure 31 - Balise. (RailSystem, n.d.)

- **Communications-Based Train Control** - CBTC is a railway signaling system that makes use of the telecommunications between the train and railway equipment for the traffic management and infrastructure control. Through CBTC systems, the exact position of a train is known more accurately than with traditional signalling systems. This results in a more efficient and safer way to manage rail traffic. (RailSystem, n.d.)

CBTC benefits include (Bombardier_C, n.d.):

- ✓ Integrated system based on state-of-the-art CBTC with bi-directional radio communication.
- ✓ Improved transport capacity to optimize use of the infrastructure: track and trains.
- ✓ Enhanced system supervision and control from centralized automatic train control (ATC) location, thanks to bi-directional communication capabilities and integrated data network
- ✓ Reduced lifecycle and maintenance costs, including the result of less wayside equipment being required.

- **ATC** – Automatic Train Control refers to a system that includes several systems (modern concept to use automatic trains). This system is (The Railway Technical_A, n.d.) :
 - **ATP** (Automatic Train Protection).
 - **ATO** (Automatic Train Operation).
 - **ATS** (Automatic Train Supervision).

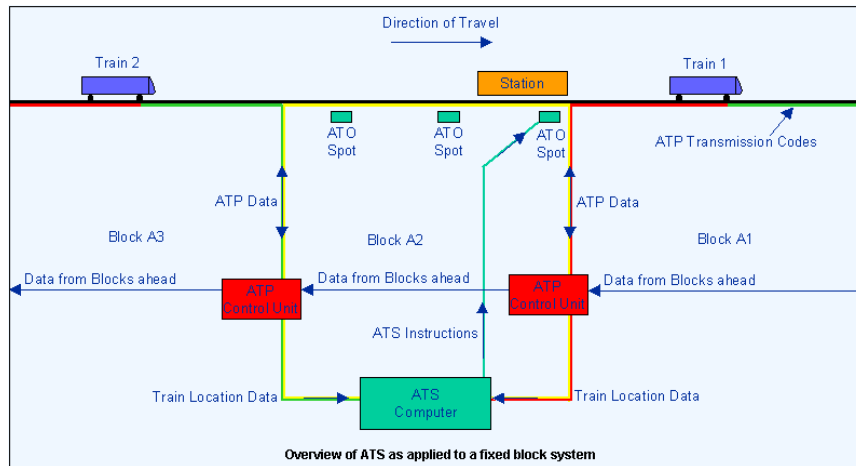


Figure 32 - A schematic showing the basic architecture of a fixed block ATC system with its three main components.
(The Railway Technical_A, n.d.)

- **ATP** – Safety system monitors against the current permitted speed limit. If the allowable speed is exceeded, a brake application is invoked until the speed is brought within the required limit or until the train is stopped. (Connor & Schmid, n.d.)

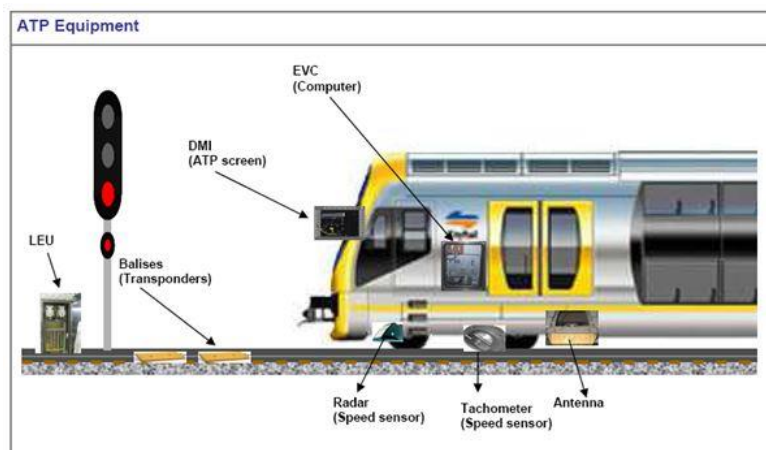


Figure 33 - ATP System (Railway Signalling Concepts, 2019)

3.2. Standards

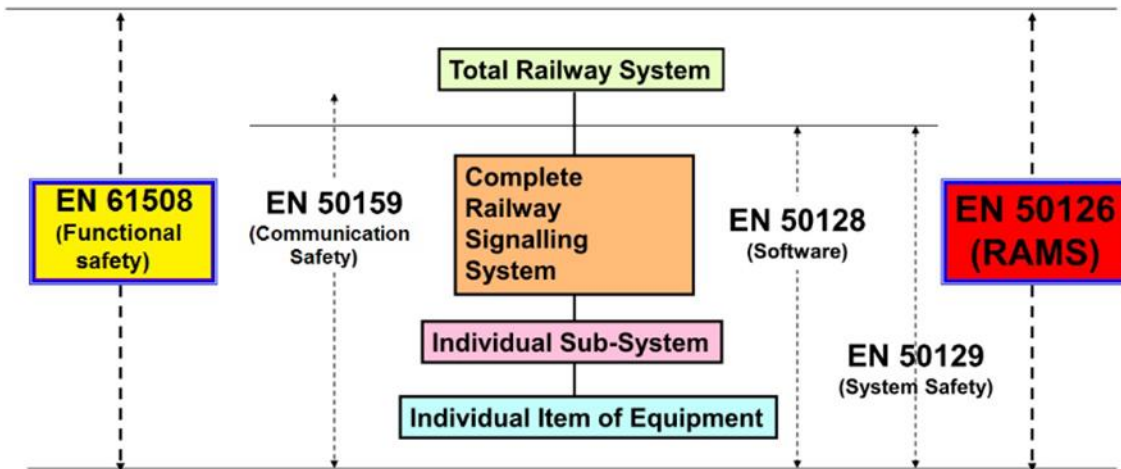


Figure 34 - Standards for Railway Application. (Wollny, 2017)

3.2.1. EN50126/8/9

CENELEC standards EN 50126, 50128 and 50129 provide guidance on RAMS (Reliability, Availability, Maintainability and Safety) for the rail industry, in Europe and further afield. (Ricardo, 2015)

3.2.1.1. EN50126

EN50126 (“Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)”) contemplates the introduction of the application of a systematic RAMS management process in the railway sector. Through the application of this standard and the experiences acquired in recent years by CENELEC, there was a need for a systematic and coherent approach to RAMS applicable to all fields of railway application (CENELEC, 2017):

- Command.
- Control and Signalling (Signalling).
- Rolling Stock.
- Electric power supply for Railways (Fixed Installations).

Examples of EN 50126 utility (Wollny, 2017):

- Identify influence factors to RAMS of a railway system.
- Manage those influence factors, i.e., evaluate the effect of each factor at each phase of the life cycle.
- Perform a risk analysis for various phases of the system life cycle and link tasks to the authority responsible.
- Structure a system life cycle for the purpose of planning, managing, controlling, and monitoring all aspects of a system, including RAMS, to deliver the right product at the right price within the agreed time scales.
- Support an audit process and to provide a basis for the railway authority and the railway support industry to agree and implement an audit plan for the railway system.

3.2.1.1.1. System Life Cycle in “V” Presentation

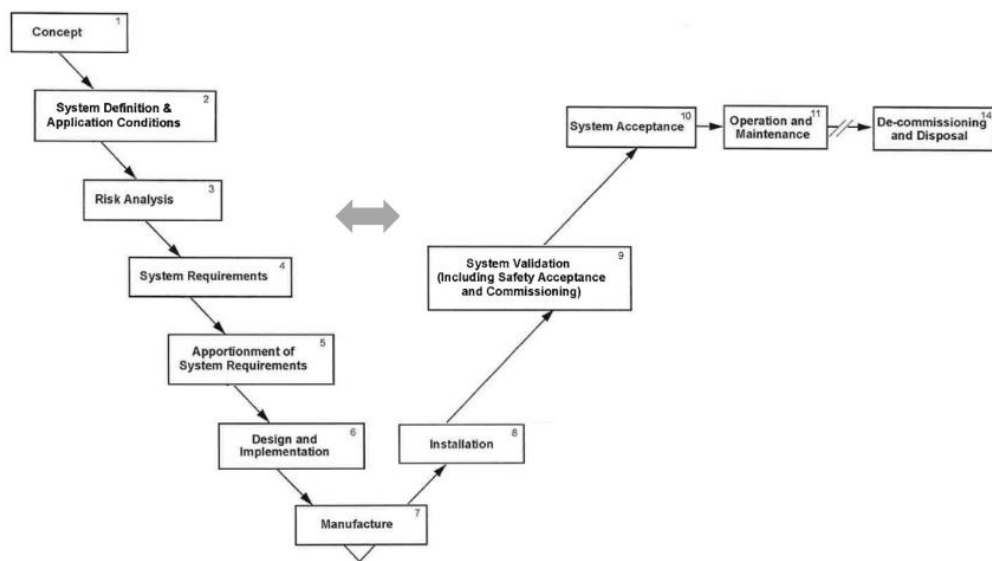


Figure 35 - System Live Cycle in "V" Presentation (Wollny, 2017) (CENELEC, 2017)

Figure 35 represents the life of the equipment, since it's development, until the end of the unit of process within the manufacturing process. The "V" representation assumes that acceptance phases are linked to the development phases: what is designed must be checked regarding requirements.

Validation activities for acceptance should be planned in the earlier stages because validation and acceptance are based on the system specification. (Wollny, 2017)

This typology will also be applied to software development, as we will see next.

3.2.1.2. EN50128

EN50128 (“Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems”) provides a set of requirements within the development, deployment and maintenance of any safety-related software intended for railway control and protection in which applications shall comply. It defines requirements concerning organizational structure, the relationship between organizations and division of responsibility involved in the development, deployment, and maintenance activities. (CENELEC, 2011)

Examples of EN 50128 utility (CENELEC, 2011):

- Define the Software Requirements Specification and in parallel consider the software architecture. The software architecture is where the safety strategy is developed for the software and the software safety integrity level.
- Design, develop and test the software according to the Software Quality Assurance Plan, software safety integrity level and the software lifecycle.
- Integrate the software on the target hardware and verify functionality.
- Accept and deploy the software.
- If software maintenance is required during operational life, then re-activate EN as appropriate.

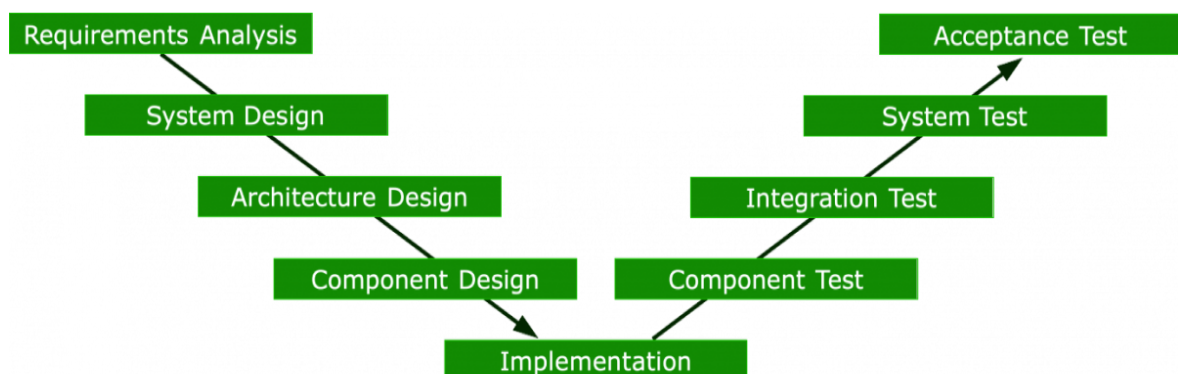


Figure 36 - V- model for SW. (Turck, 2020)

3.2.1.3. EN50129

EN50129 (“Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling”) applies to systems for generic or for specific applications, in this case for the functional safety of systems. This standard presents all the phases of the life cycle of a safety-related electronic system, focusing architecture and apportionment of system requirements to system acceptance. (CENELEC, 2018)

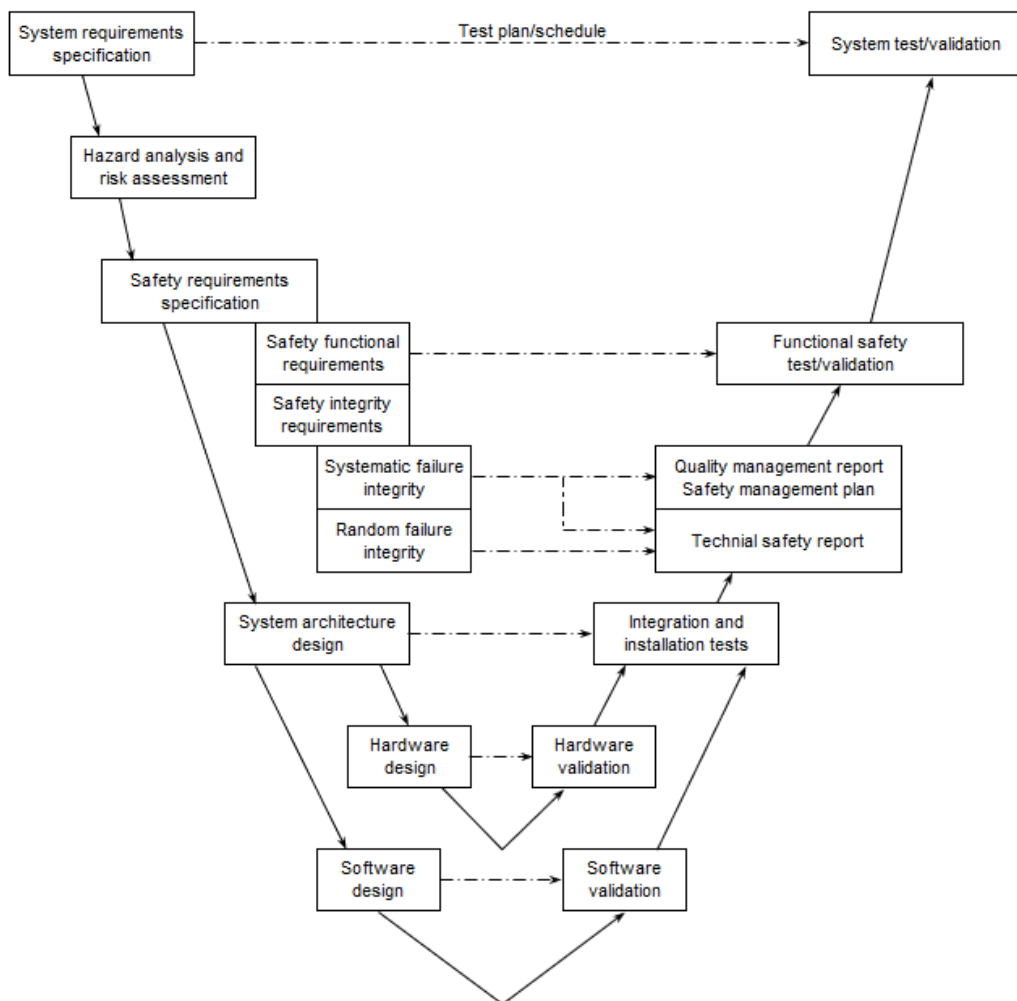


Figure 37 - Model V for EN50129 (Li, 2019)

3.2.2. IEC 61508

IEC 61508 is an international functional safety standard, consisting of methods on how to apply, design, deploy and maintain automatic protection systems called safety-related systems (IEC, n.d.). Based on this standard, EN 50128 was created, dedicated to Railway.

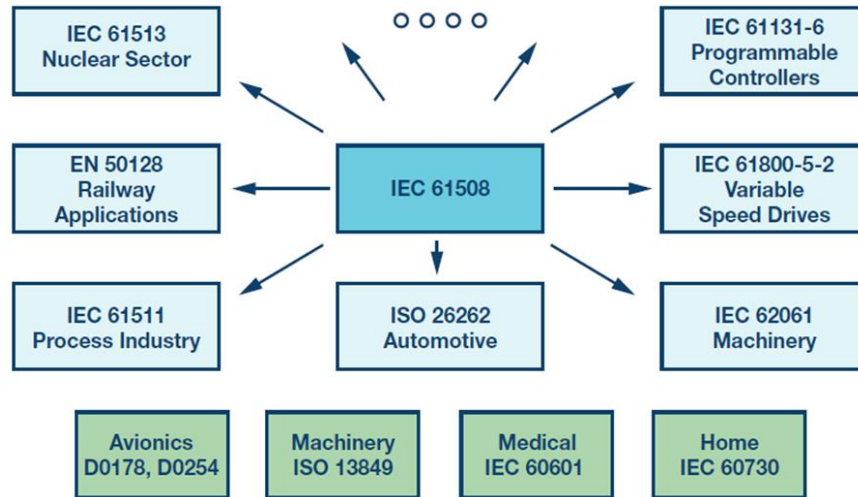


Figure 38 - IEC 61508 derived documents. (AHMAD, 2020)

Some definitions:

- **Functional safety:** is the detection of a potentially dangerous condition resulting in the activation of a protective or corrective device or mechanism to prevent hazardous events arising or providing mitigation to reduce the consequence of the hazardous event. (IEC, n.d.)
- **Safety Integrity:** is defined as “The probability of a Safety Instrumented Function (SIF) satisfactorily performing the required safety functions under all stated conditions within a stated period of time”. (Borges, 2017)
- **Safety Integrity Level:** it is a discrete level (one out of a possible four) for specifying the safety integrity requirements of the safety functions to be allocated to the safety-related systems. (Borges, 2017)

3.2.2.1. Safety Integrity Level (SIL)

Table 2 - Safety Integrity Level (Borges, 2017)

Safety Integrity Level	Probability of Failure on Demand	Risk Reduction Factor
SIL 4	$\geq 10^5$ to $< 10^4$	100,000 to 10,000
SIL 3	$\geq 10^4$ to $< 10^3$	10,000 to 1,000
SIL 2	$\geq 10^3$ to $< 10^2$	1,000 to 100
SIL 1	$\geq 10^2$ to $< 10^1$	100 to 10

Frequency	5	SIL3	SIL4	X	X	X
	4	SIL2	SIL3	SIL4	X	X
	3	SIL1	SIL2	SIL3	SIL4	X
	2	-	SIL1	SIL2	SIL3	SIL4
	1	-	-	SIL1	SIL2	SIL3
		1	2	3	4	5
Severity of Consequence						

Figure 39 - Resume of Table of SIL (Cross, n.d.)

3.2.3. DO178C

DO-178C (“Software Considerations in Airborne Systems and Equipment Certification”) is a standard that approves all commercial software-based aerospace systems (Brosgol & Comar, 2010).

Objectives of this standard, are the following:

- Promote safe implementation of aeronautical software.
- Provide clear and consistent ties with the systems and safety processes.
- Address emerging software trends and technologies.
- Implement an approach that can change with the technology.

3.2.3.1. Safety Level

This standard induces safe critical of an aircraft. For aircraft and for any other equipment that involves safety, Safety Level is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system. The failure conditions are categorized by their effects on the aircraft, crew, and passengers. (Wikipedia_H, 2020).

The classification of the levels is the following (Brosgol & Comar, 2010):

- **Level A (Catastrophic)** - Failure may cause deaths, usually with loss of the airplane.
- **Level B (Hazardous)** - Failure has a large negative impact on safety or performance or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload or causes serious or fatal injuries among the passengers.

- **Level C (Major)** - Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries).
- **Level D (Minor)** - Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change.
- **Level E (No Effect)** - Failure has no impact on safety, aircraft operation, or crew workload.

These levels consider SIL that were already referred, however for aircraft it has another classification (i.e. DAL), nevertheless they are the same.

Engineering “Independence” per Development Assurance Level (DAL). (Note Quality Assurance must always be independent):

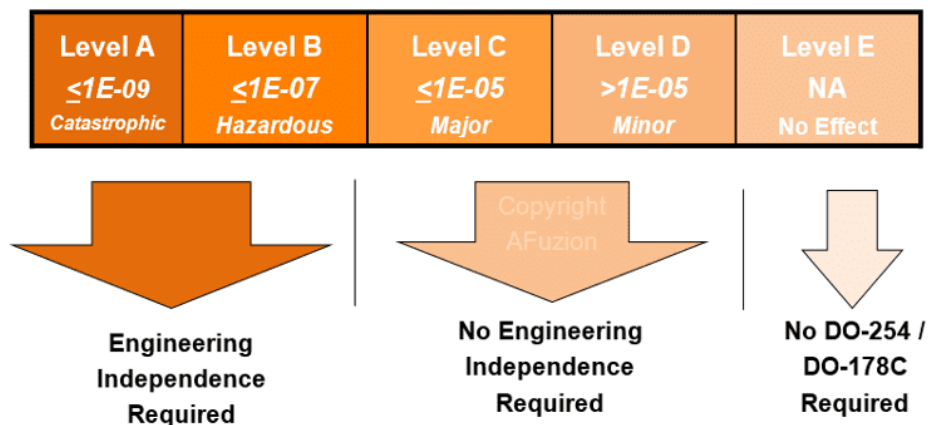


Figure 40 - Safety Level in DO178C. (Hilderman, 2014)

3.2.4. ISO 26262

ISO 26262 is an international standard for functional safety in the automotive industry. The standard applies to electrical and electronic systems consisting of hardware and software components in vehicles. It defines requirements to be met by the safety relevant function of the system as well as by processes, methods and tools which are used within the development process. The ISO 26262 standard ensures that sufficient levels of safety are being met and maintained throughout the vehicle lifecycle. (TÜV SÜD, n.d.)

Objectives, of this standard, are the following:

- Demonstrate due diligence and ensure overall safety of the respective vehicle.
- Maintain your competitive advantage in interpreting and implementing the standard requirements correctly.
- Minimize the risk of harm to people and non-acceptance of the products manufactured.
- Avoid costly product recalls and reputational damage due to safety hazards because insufficient safety assurance.
- Simplify access to global markets by ensuring compliance with relevant international regulations.

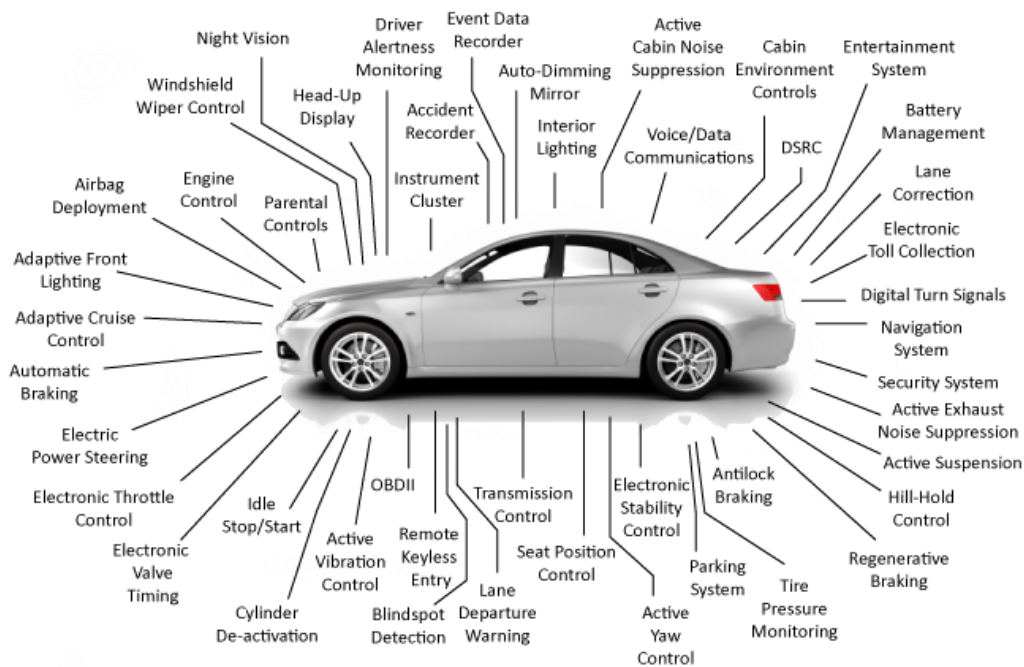


Figure 41 - Systems in Car. (Mannion, 2017)

3.2.5. ISO/IEC62443

The standard IEC 62443 (Industrial communication networks - Network and system security) certifies networks use in automation industry, covering both organizational and technical aspects of security over the life cycle of systems.

This standard is divided in four parts (IEC, n.d.):

- The General parts
 - defines the terminology, concepts and models for Industrial Automation and Control Systems.
 - gives the definition of terms and acronyms.
- The Policies and Procedures related parts
 - specifies asset owner security program requirements for an industrial automation and control systems, it also provides guidance on how to develop and evolve the security program.
 - specifies a framework and methodology for evaluation of the protection of an Industrial Automation and Control Systems (IACS) based on the notion of security level and the maturity of the connected processes.
 - defines the patch management in the IACS environment.
 - specifies requirements for security capabilities for IACS service providers that they can offer to the asset owner during integration and maintenance activities of an Automation Solution.
- The System related parts
 - provides a current assessment of various cybersecurity tools, mitigation countermeasures, and technologies that may effectively apply to the modern electronically.
 - establishes requirements for risk assessment to partition an IACS into zones and conduits.
 - provides detailed technical control system requirements associated with the seven foundational requirements, including defining the requirements for control system capability security levels.
- The Component related parts
 - specifies process requirements for the secure development of products used in industrial automation and control systems.
 - specifies the cyber security technical requirements for components, such as embedded devices, network components, host components and software applications.

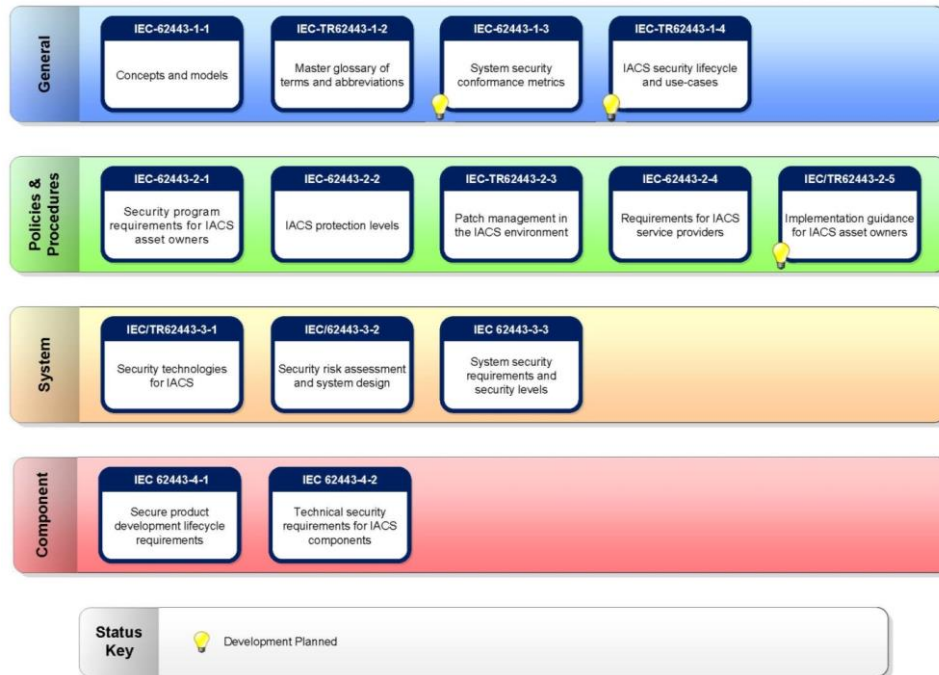


Figure 42 - IEC 62443 Series of Industrial Security Standard. (IEC, n.d.)

3.2.6. ISO 27001

ISO 27001 provides requirements for an information security management system. This standard enables organizations of any kind to manage the security of assets such as financial information, intellectual property, employee details or information entrusted by third parties. (ISO, n.d.)



Figure 43 - Process of ISO 27001. (XpertLync, n.d.)

The benefits of working with standard (ISO 27001) (Imperva, n.d.) are seen in different areas, such as:

- **Risk management** – An Information Security Management (ISMS) helps govern who within an organization can access specific information, reducing the risk that said information can be stolen or otherwise compromised.
- **Information security** – An ISMS contains information management protocols detailing how specific data needs to be handled and transmitted.
- **Business continuity** – To remain ISO 27001 compliant, a service provider's ISMS must be continuously tested and improved upon. This helps prevent data breaches that could impact your core business functions.

3.3. Testing techniques

3.3.1. Functional Testing

Functional Testing is a type of software testing, by which the system is tested against the functional requirements/ specifications. (Functional Testing, 2020)

Code (Functions) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied.

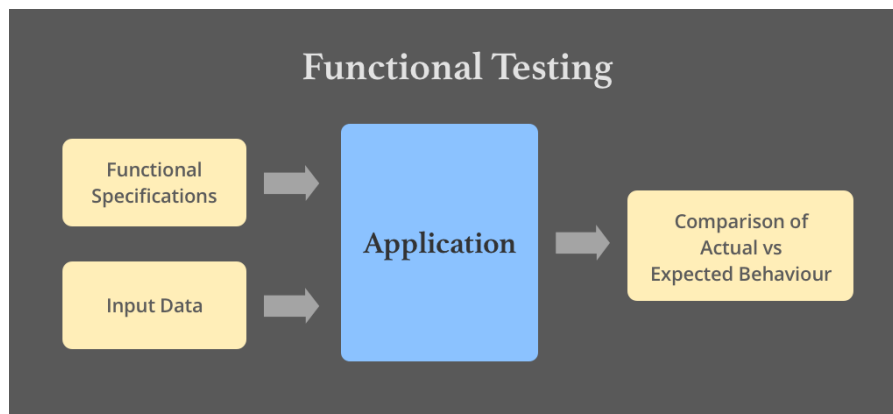


Figure 44 - Example of Functional Testing. (Shah, n.d.)

Steps to do Functional Testing (Wikipedia_J, 2020) (Guru99_C, n.d.):

1. The identification of functions that the software is expected to perform.
2. The creation of input data based on the function's specifications.
3. The determination of output based on the function's specifications.
4. The execution of the test case.
5. The comparison of actual and expected outputs.
6. To check whether the application works as per the requirement do.

3.3.1.1. Types of Functional Testing

There are several types of Functional Testing. Therefore, we will present the ones that are most considered by publications:

- **Unit Testing** – Testing of units/components of software, without integrating it with other components.
- **Smoke Testing** – Testing of software build to ensure that the critical functionalities of the program are working fine.
- **Sanity Testing** - Testing performed after a minor software build/patch, with few changes in code/ functionality like a bug fix.
- **Integration Testing** - Testing where individual units are consolidated and tested as a group.
- **User Acceptance** - Testing an application from the end-user or the client perspective to verify the usability and accessibility of the software system before moving to the production environment.
- **Regression Testing** - testing approach to confirm that a new code change has not affected existing features.

3.3.2. Black Box Testing

Black Box Testing is a software testing method, that focus on the functionalities of the software applications that are tested without having knowledge of the internal code structure (implementation details and internal paths). Black Box Testing focuses on input and output of software applications and analysis based on software requirements and specifications. (Behavioral Testing) (Guru99_B, n.d.)

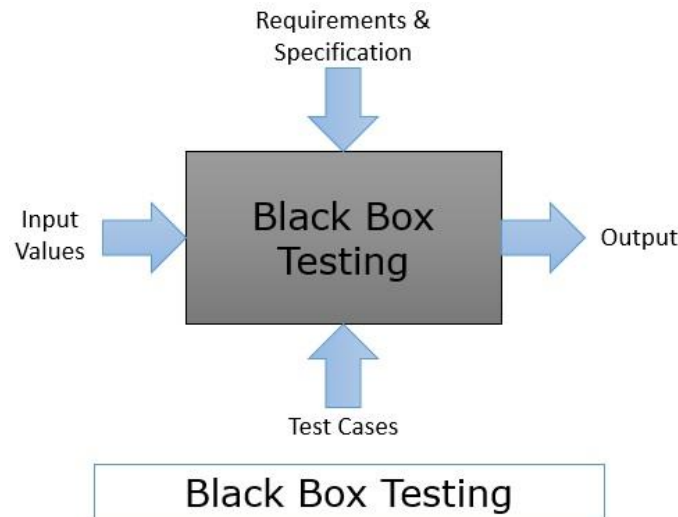


Figure 45 – Black-box Testing. (T, 2020)

There are many types of Black Box, but in this project, we will use Functional Testing, which has already been introduced above.

Steps to Black Box Testing (Guru99_B, n.d.) :

1. First the requirements and specifications of the system are examined.
2. Then the tester chooses valid inputs (positive test scenario) to check if they are being processed correctly.
3. Following the tester determines expected outputs for all those inputs.
4. Later the software tester constructs test cases with the selected inputs.
5. Next the test cases are executed.
6. After the software tester compares the actual outputs with the expected outputs.
7. Finally, it detects if there is any error.

3.3.2.1. Techniques of Black Box Testing

According to the signals / values we have input and output, that can be used in several data analysis techniques. Some examples (QATestLab, 2018) (Guru99_B, n.d.):

- **Boundary value** - It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
- **Equivalence class** - Testing based on checking one value within one input class.
- **Error guessing** - Based on the previous experience.

- **Decision table testing (MC/DC)** - Based on tabular representation of combinations of inputs and correspondent system behavior.
- **Graph based testing** - Where a test case is written for each graph that represents the object.

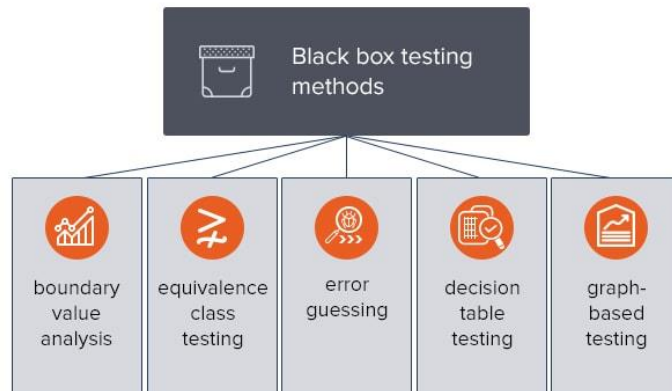


Figure 46 - Black box testing. (QATestLab, 2018)

3.3.3. White Box Testing

White Box Testing is a software testing method in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability, and security. In white box testing, code is visible to testers. (Guru99_D, n.d.)

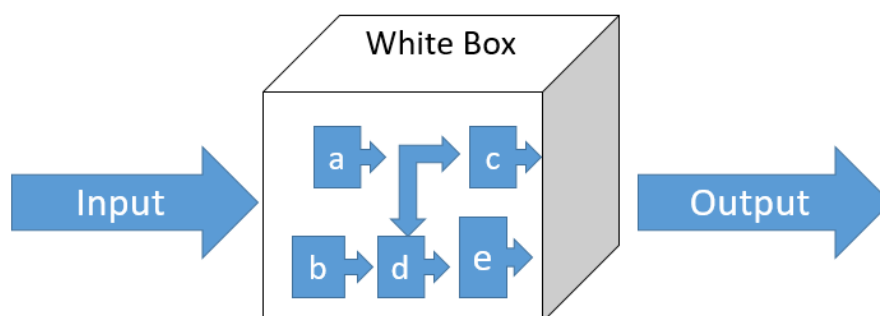


Figure 47 - White Box Testing. (Dobra, 2019)

Steps to White Box Testing (Johnson, 2020) (Wikipedia_G, 2021):

1. Input involves different types of requirements, functional specifications, detailed designing documents of project, source code and security specifications. This step presents all information of testing in question.
2. Processing involves performing risk analysis to guide the whole testing process, appropriate test plan, execute test cases and communicate results. This step of building test cases is used to make sure they thoroughly test the application, and the given results are recorded accordingly.
3. Output involves preparing a final report that includes all the above preparations and results.

3.3.3.1. White Box Verification

White box testing involves the testing of the software code for the following (Guru99_D, n.d.):

- Internal security holes.
- Broken or poorly structured paths in the coding processes.
- The flow of specific inputs through the code.
- Expected output.
- The functionality of conditional loops.
- Testing of each statement, object, and function on an individual basis.

3.3.3.2. Techniques of White Box Testing

Introducing Main White Box Techniques (Guru99_D, n.d.):

- **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.
- **Branch Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.

- **Loop Testing:** Loops are widely used, and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginning and end of loops.

Other types of tests can also be considered, such as:

- Decision Coverage
- Condition Coverage
- Multiple Condition Coverage
- Finite State Machine Coverage
- Path Coverage
- Control flow testing
- Data flow testing

3.3.4. Differences between Black box vs White Box Testing

Box Testing can be divided in two types, as already seen before. One of the types, is Black Box testing, which involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The table below shows differences between White Box and White Box.

Table 3 - Differences between Black Box Testing vs White Box Testing. (Jain, 2020)

Black Box Testing	White Box Testing
It is a way of software testing in which the internal structure or the program or the code is hidden, and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated based on requirement specifications document.	This type of testing of software is started after detail design document.

No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It consumes less time.	It consumes more time.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial-and-error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	Example: by input to check and verify loops
Types of Black Box Testing: A. Functional Testing B. Non-functional testing C. Regression Testing	Types of White Box Testing: A. Path Testing B. Loop Testing C. Condition testing

3.3.5. Static program analysis

Static program analysis is a method of computer program debugging that is done by examining the code without running the program. This process provides an understanding of the code structure and can help ensure that the code meets the industry standards. Static analysis is used in software engineering by software development and quality assurance teams. Automated tool can assist programmers and developers in carrying out static analysis. (Rouse, n.d.) (Dietrich, n.d.)

This analysis is good to identify coding issues such as:

- Programming errors.
- Coding standard violations.
- Undefined values.
- Syntax violations.
- Security vulnerabilities.



Figure 48 - Flow Static Analysis. (Mezquita, 2020)

3.4.What is Component Testing?

This test (Component Test) is often performed isolated from the rest of the system, depending on the software development lifecycle model and the system. (International Software Testing Qualifications Board, 2019)

Component testing may cover, non-functional characteristics and structural properties. The objectives of component testing include (International Software Testing Qualifications Board, 2019):

- ✓ Reducing risk.
- ✓ Verifying whether the functional and non-functional behaviors of the component are as designed and specified.
- ✓ Building confidence in the component's quality.
- ✓ Finding defects in the component.
- ✓ Preventing defects from escaping to higher test levels.

Table 4 - Table of Resume of Component Testing.

Test basis:	Test objects:	Typical defects and failures detect:
<ul style="list-style-type: none"> ✓ Detailed design ✓ Code ✓ Data model ✓ Component specifications 	<ul style="list-style-type: none"> ✓ Components, units, or modules ✓ Code and data structures ✓ Classes ✓ Database modules 	<ul style="list-style-type: none"> ✓ Incorrect functionality ✓ Data flow problems ✓ Incorrect code and logic

3.5. What is Continuous Integration and Continuous Delivery (CI/CD)?

CI/CD (Continuous Integration and Continuous Delivery) is a method to regularly deliver software changes in the process of development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code in software development. (Red Hat, n.d.)

CI/CD introduces automation and continuous monitoring throughout the lifecycle of Project, from integration and testing phases to delivery and deployment.

3.5.1. What is the difference between CI and CD?

In a few words, CI (Continuous Integration) implies that every time there is a new code different from the source code there will be a new build, that will be tested, and merged to a shared repository. In contrast, a CD (Continuous delivery) approach is where the software can be deployed to a live production environment by the operations team. In some reference's CD (Continuous deployment) represents a methodology that allows automatically release according to developer's changes from the repository to production, where it is usable by customers. (Red Hat, n.d.)



Figure 49 - Representation of CI/CD (Red Hat, n.d.).

3.5.2. Continuous Integration

Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. Allowing developers to frequently merge code changes into a master repository where builds are being tested. Automated tools are used to assert the new code's correctness before integration. (Rehkopf, n.d.)

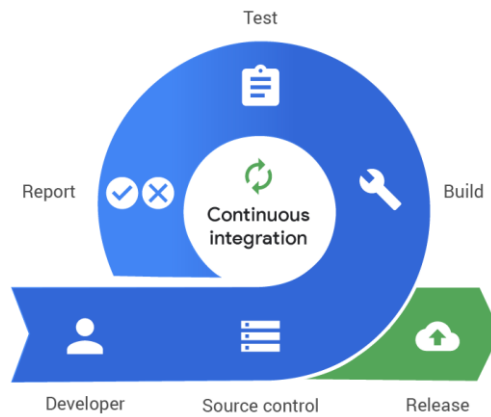


Figure 50 - Continuous Integration. (PagerDuty, n.d.)

3.5.3. Continuous Delivery

Continuous delivery (CD) is an extension of continuous integration. Developers practicing continuous integration merge their changes back to the main branch as often as possible to make sure that they can release new changes more quickly and in a sustainable way. This means that on top of having automated testing and release process, the product can be deployed at any time. (Pittet, n.d.)

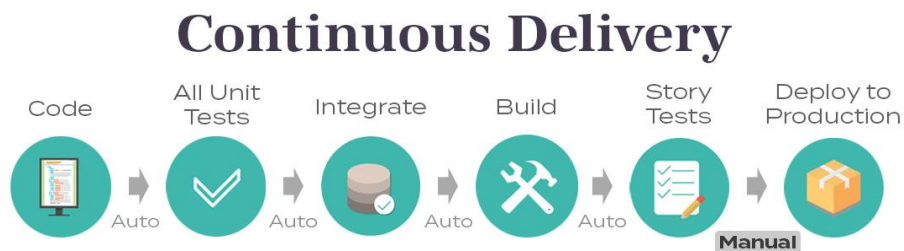


Figure 51 - Continuous Delivery. (ATC Team, 2019)

4. Workflow in CSW Project

4.1. Introduction

As already mentioned, this work is based on the component test, therefore, the objective will be to carry out tests to verify the validity of the software regarding the requirements of both the customer and the standards.

This analysis will be performed considering that we have a black box (as already seen, we do not know what it is inside), and to know if it behaves properly, we will have to analyze the inputs and outputs, which helps creating impartiality over existing code. White Box testing can also be used for tests that must be carried out as part of the process, for the interpretation of some code parts that, if we have only the requirements can be ambiguous. This is necessary to test (Pre-conditions) or find problems in test.

In this chapter, I will show you all the steps that must be performed, from understanding the function of each component, to performing the tests.

Steps:

- Analysis
- Design
- Implementation
- Execution and Report

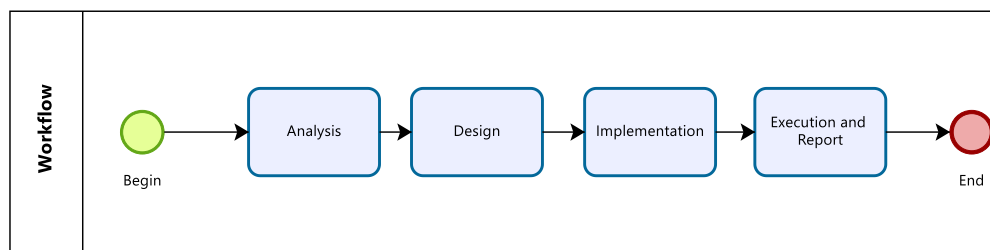


Figure 52 - Workflow Steps.

A more simplified version of the process can be represented as follows:

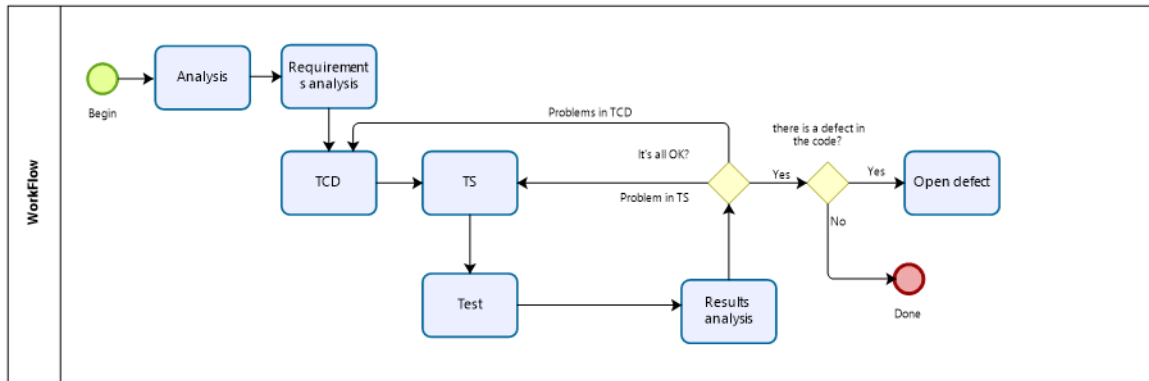


Figure 53 - Workflow in Project Summary.

4.2. Analysis

First step of the workflow is to analyze the problem in our hands (Functional Requirement (FR) / Change Request (CR) / Defect). The objective is to verify which TCD (Test Cases Design) already exists, if they need to be changed, and in case it does not exist, new TCDs need to be created. Some SRSs can be tested together in the same TCD, this is the phase that will be checked.

Some topics of what was done in this phase:

- Analysis of requirements in documents in analysis.
- Verify status of requirements, and history.
- Check condition of already made TCDs. Run TS (Test Script) and check TCD, and check still fits requirement, if not re-open the Test Case (to remake).
- Comply with the formalities for creating new TCDs.

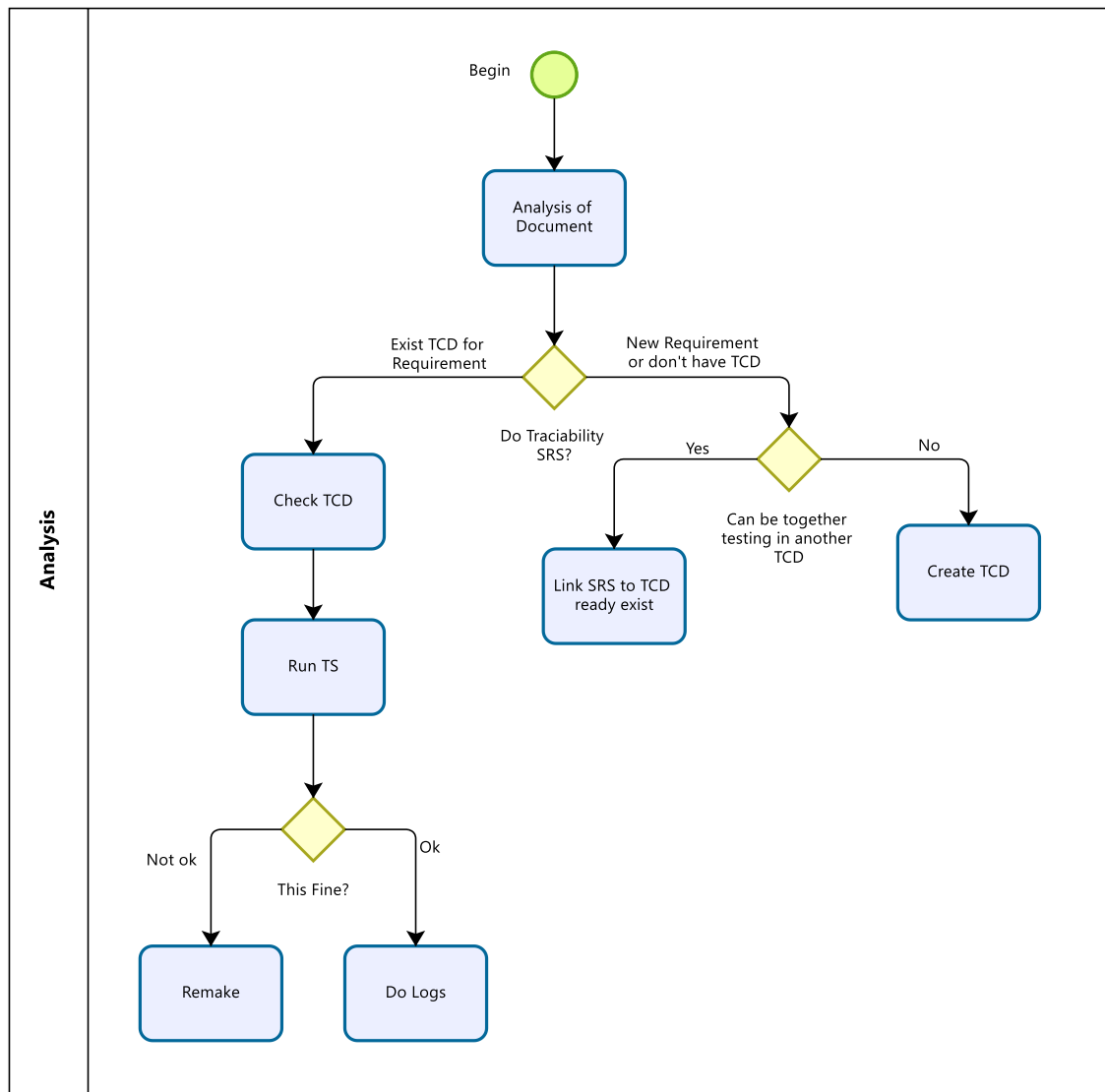


Figure 54 - Workflow in analysis phase.

4.2.1. Requirements analysis

The requirements analysis is an essential part of this work, from here it will be possible to check how the tests are carried out.

From this analysis, we will have the indication of which signals have to be analyzed. For example, from the Requirement provided for analysis (unreal example):

Table 5 - Example of Requirement

Given	Command Emergency Brake is SET
When	Emergency brake loop is energized

Then	Fault is sent to control unit
------	-------------------------------

These requirements will also indicate the safety level (SIL), which will give an insight about the response time, for example.

4.2.1.1. Glossary

Table 6 - Table Glossary of Requirement.

Clauses	Definition
Given	Define the Pre-Conditions below which the When and Then clause apply.
When	Define the action to do (trigger) that will trigger the expected output.
Then	Define the expected result of output of the requirement.

- **Given-When-Then**

Changes in the Given or in the trigger after obtaining the result, do not affect the result unless specified by another requirement. For example, for this type of requirement we need a requirement of set and another for reset. While with the given-then requirement if a simple condition fail the output, signal is reset.

4.3.Design

In this design phase, we will focus more on process analysis and requirement analysis. For that, it will be necessary to access some documents.

For example, we can access the Interface Signal List (ISL) and MIO List, to know which signals we have at our disposition to fulfill the requirement, and if necessary, we must analyze the schematic, as a help in the knowledge of functionality.

For general knowledge of the functionality, some knowledge came from the previous phase. Is also important to have technical references of the functionality, for that it is necessary to use FVDS and other documents of CSW client.

In these documents/research we may also find data on values, such as valid and invalid values, values for default, among other relationships between signals.

Then the techniques already referred were applied to obtain a robust Test Case Design (TCD) that in most cases meet the requirements.

After completing the TCD we will be ready to create the Test Script (TS) and send TCD for necessary approvals.

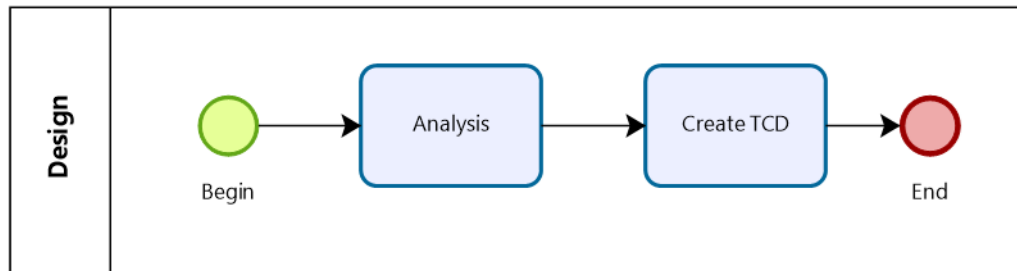


Figure 55 - Design Step.

4.3.1. Test Case Design

In the Test Case Design (TCD) there will be indicated prerequisites, actions to be done and post executions.

4.3.1.1. Structure of TCD

- Pre-conditions are applied to all signals or states that the system must have before the test case can be applied.
- Actions will be applied to the necessary signals (force signals to their necessary value).
- Expected Result is an expected result at the exit (output), which we want to check.
- Post-conditions are the exit conditions (that must be done before finishing the test).

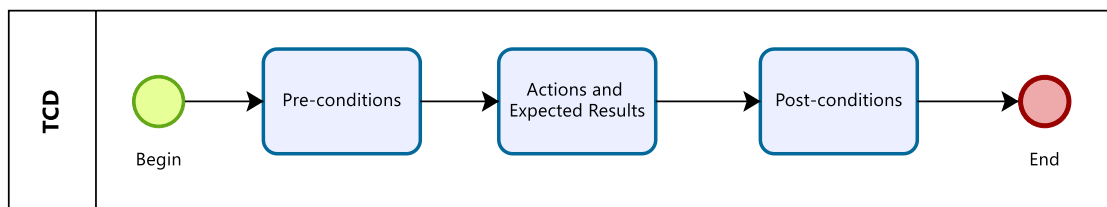


Figure 56 - TCD Components.

```
-----  
PRE-CONDITION  
Condition before run actions  
Force ...  
-----  
TEST CASE DESIGN  
  
Action 1: ...  
1. Set ...  
.  
.  
.  
Expected result:  
1. Verify ...  
...  
.  
.  
  
Action 2:  
...  
.  
.  
  
-----  
POST-CONDITION  
Condition after Run  
Unforce|
```

Figure 57 - Example of Test Case Design.

4.3.2. Example TCD

- Requirement:
 - Requirement 1:

Table 7 - Requirement 1.

Given	The table for fail-safe reaction evaluation (Table verification of state) AND this is the active cab AND train configuration is confirmed
When	A push button from the considered cab becomes set for at least a threshold time for the button to be pressed
Then	CCU-S shall set the output command for the respective push button as a pulse having a period of a parametrizable time

- Requirement 2:

Table 8 - Requirement 2.

Given	A push button is invalid/inconsistent according to the table (Table verification of state)
Then	CCU-S shall set the output command for the respective push button

- TCD Realized:

- Preconditions:

- | |
|--|
| <ol style="list-style-type: none"> 1. Force Train Configuration is valid to TRUE 2. Verify that Close Doors PB threshold time is 2 seconds 3. Verify that Close Doors PB command pulse time to 1 second |
|--|

- Actions:

Table 9 - Table of Actions Realized for Requirement present.

<ul style="list-style-type: none"> • Repeat all actions for cab number <CAB_NUMBER> • Repeat all actions for pulse time <PULSE_TIME> • Repeat all actions for each Combination Active <COMBINATION_ACTIVE> • Repeat actions 1 to 10 for threshold time <THRESHOLD_TIME> • Repeat action 1 and 11 for cab state <CAB_STATE> • Repeat action 4 and 13 for each Combination Inactive <COMBINATION_INACTIVE> • Repeat action 15 and 16 for each Combination Inconsistent <COMBINATION_INCONSISTENT>
<p>Actions:</p> <p>Action 01: Verify Close Doors PB command is reset when combined cab state is inactive</p> <ol style="list-style-type: none"> 1. Force Close Doors PB threshold time to <THRESHOLD_TIME> 2. Force Close Doors PB command pulse time to <PULSE_TIME> 3. Force combined cab state to <CAB_STATE> 4. Force Train Configuration Confirmed to TRUE 5. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> + 150ms on cab <CAB_NUMBER> <p>Expected Results:</p> <ol style="list-style-type: none"> 1. Verify Close Doors PB command is reset on both cabs and General Signal for 1000ms

Action 02: Verify Close Doors PB command is reset when combined cab state is active after the Close Doors PB is set for more than <THRESHOLD_TIME>

1. Force Close Doors PB input combination to <COMBINATION_ACTIVE> on cab <CAB_NUMBER>
2. Force combined cab state to <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset for <THRESHOLD_TIME> on both cabs and General Signal
2. Verify Close Doors PB command is set for <PULSE_TIME> on <CAB_NUMBER> and General Signal

Action 03: Verify Close Doors PB command is reset when Close Doors PB is reset

1. Force Close Doors PB input combination to Inactive combination on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 04: Verify Close Doors PB command is reset when Close Doors PB is reset

1. Force Close Doors PB input combination to <COMBINATION_INACTIVE> for <THRESHOLD_TIME> + 150ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 05: Verify Close Doors PB command is set when combined cab state is active and Close Doors PB is set for more than <THRESHOLD_TIME>

1. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> + 150ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is set for <PULSE_TIME> on <CAB_NUMBER> and General Signal

2. Verify Close Doors PB command is reset on both cabs and General Signal

Action 06: Verify Close Doors PB command is set when combined cab state is active and Close Doors PB is set for <THRESHOLD_TIME>

1. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is set for <PULSE_TIME> on <CAB_NUMBER> and General Signal

2. Verify Close Doors PB command is reset on both cabs and General Signal

Action 07: Verify Close Doors PB command is reset when Train Configuration Confirmed is FALSE

1. Force Train Configuration Confirmed to FALSE

2. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> + 150ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 08: Verify Close Doors PB command is reset when Train Configuration is valid is FALSE

1. Force Train Configuration is valid to FALSE

2. Force Train Configuration Confirmed to TRUE

3. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> + 150ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 09: Verify Close Doors PB command is reset when Train Configuration is valid is TRUE

1. Force Train Configuration is valid to TRUE

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 10: Verify Close Doors PB command is reset when combined cab state is active and Close Doors PB is set for less than <THRESHOLD_TIME>

1. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for <THRESHOLD_TIME> - 200ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

Action 11: Verify Close Doors PB command is reset when combined cab state is inactive

1. Force Close Doors PB threshold time to 0

2. Force combined cab state to <CAB_STATE>

3. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for 150ms on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

<p>Action 12: Verify Close Doors PB command is set when combined cab state is active after the Close Doors PB is set for 0 seconds</p> <ol style="list-style-type: none"> 1. Force Close Doors PB input combination <COMBINATION_ACTIVE> on cab <CAB_NUMBER> 2. Force combined cab state to <CAB_NUMBER> <p>Expected Results:</p> <ol style="list-style-type: none"> 1. Verify Close Doors PB command is set for <PULSE_TIME> on <CAB_NUMBER> and General Signal 2. Verify Close Doors PB command is reset on both cabs and General Signal
<p>Action 13: Verify Close Doors PB command is reset when Close Doors PB is reset</p> <ol style="list-style-type: none"> 1. Force Close Doors PB input combination to <COMBINATION_INACTIVE> for 150ms on cab <CAB_NUMBER> <p>Expected Results:</p> <ol style="list-style-type: none"> 1. Verify Close Doors PB command is reset on both cabs and General Signal
<p>Action 14: Verify Close Doors PB command is set when combined cab state is active and Close Doors PB is set for more than 0 seconds</p> <ol style="list-style-type: none"> 1. Force Close Doors PB input combination to <COMBINATION_ACTIVE> for 200ms on cab <CAB_NUMBER> <p>Expected Results:</p> <ol style="list-style-type: none"> 1. Verify Close Doors PB command is set for <PULSE_TIME> on <CAB_NUMBER> and General Signal 2. Verify Close Doors PB command is reset on both cabs and General Signal
<p>Action 15: Verify Close Doors PB command is set when Close Doors PB is Inconsistent</p> <ol style="list-style-type: none"> 1. Force Close Doors PB input combination to <COMBINATION_INCONSISTENT> on cab <CAB_NUMBER> <p>Expected Results:</p> <ol style="list-style-type: none"> 1. Verify Close Doors PB command is set on General Signal

Action 16: Verify Close Doors PB command is reset when Close Doors PB is reset

1. Force Close Doors PB input combination to Inactive combination on cab <CAB_NUMBER>

Expected Results:

1. Verify Close Doors PB command is reset on both cabs and General Signal

○ Post-Conditions:

Unforce all signals

4.4.Implementation

In this phase of the process, it is done everything related with the implementation, first execution of the test case design and identification of defects.

For this, sequences of steps for TS operation must be carried out:

- Test Case Design has to be completed.
- Implementation of the functions needed to force signaling (when the first time it is used).
- Creation of the TS following the rules for its creation.
- Update the test logs in question.

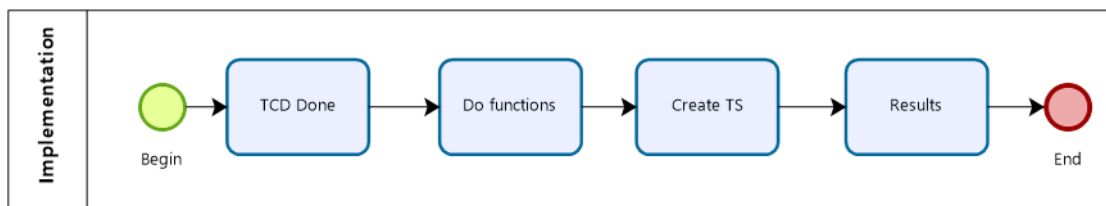


Figure 58 - Implementation step.

4.4.1. How Tests Are Performed

The CSW customer at provides a certified testing tool for testing these applications. This tool is based on C# (using Microsoft Visual (Integrated Development Environment - IDE)), it can connect to simulator of CCU, "CCUS" or "CCUO", thus is able to make simulations in real time of the system.

This tool also has a graphical panel that gives information in real time, while testing, but also creates logs that prove the same test.

As it is predictable, the objective will be the creation of scripts and functions necessary to run in the simulator.

4.4.2. Example TS

Then TS is shown already presented requisite.¹ From the example shown above for TCD, some excerpts from his own TS², will be presented.

Table 10 - Examples of code realized.

Preconditions:
<pre>// Force Train Configuration is valid to TRUE sp.Set_Train_Configuration_is_Valid(status: true);</pre>
Actions:
<pre>#region Action 02: Verify Close Doors PB command is reset when combined cab state is active after the Close Doors PB is set for more than <THRESHOLD_TIME> Trace("Action 02"); sctx.step("Verify Close Doors PB command is reset when combined cab state is active after the Close Doors PB is set for more than " + threshold); sctx.kindOfTest("Positive"); sctx.prepare("1. Force Close Doors PB input combination to " + COMBINATION_ACTIVE[i, 0] + " " + COMBINATION_ACTIVE[i, 1] + " " + COMBINATION_ACTIVE[i, 2] + " " + COMBINATION_ACTIVE[i, 3] + " on cab " + CAB_NUMBER[j]); sp.Set_Doors_Close_Input_Combinations(car: car[j], comb: COMBINATION_ACTIVE, i: i); sctx.prepare("2. Force combined cab state to " + CAB_NUMBER[j]); sp.SetCombinedCabState_Stabilize(CAB_NUMBER[j], timeStabilize: 2000);</pre>

¹ In annex, I will present some examples of this process, due to confidentiality agreements, they will only be presented on paper in the presentation day. This document presents some parts of an example.

²Due to confidentiality agreements.

```
sctx.expect("1. Verify Close Doors PB command is reset for " + threshold +
" on both cabs and General Signal");
sp.VerifyCommandDriverDesk(PB: PushButtons_Switches.Door_Close, status:
false, timetocheck: thresholdsim, multiSign: true);

sp.Verify_Command_Driver_Desk_in_Cabs(CAB_NUMBER, PB:
PushButtons_Switches.Door_Close, status: false, timetocheck:
thresholdsim);
sctx.expect("2. Verify Close Doors PB command is set for " + pulse + " on
" + CAB_NUMBER[j] + " and General Signal");
sp.VerifyCommandDriverDesk(PB: PushButtons_Switches.Door_Close, status:
true, timeout: timeOut, timetocheck: pulsesim, multiSign: true);
sp.Verify_Command_Driver_Desk_in_Cabs(cab: new CabActive[] { CAB_NUMBER[j]
}, PB: PushButtons_Switches.Door_Close, status: true, timeout: timeOut,
timetocheck: pulsesim);

#endregion
```

4.4.3. Results

With the CWS client tool it is possible to have results data, that enable them to be analyzed and to be delivered to the same client.

In the image below, excerpts from the same log of the test results I have presented³.

Table 11 - Results Table of results.

263	PASSED	2021-02-26T10:36:00.202Z	SCTX_STEP:Verify Close Doors PB command is reset when combined cab state is active after the Close Doors PB is set for more than 20000
264	PASSED	2021-02-26T10:36:00.202Z	SCTX_KINDOFTTEST:Positive
265	PASSED	2021-02-26T10:36:00.202Z	SCTX_PREPARE:1. Force Close Doors PB input combination to True False True True on cab Cab1
267	PASSED	2021-02-26T10:36:00.238Z	SCTX_PREPARE:2. Force combined cab state to Cab1
275	PASSED	2021-02-26T10:36:00.765Z	SCTX_EXPECT:1. Verify Close Doors PB command is reset for 20000 on both cabs and General Signal
278	PASSED	2021-02-26T10:36:16.919Z	SCTX_EXPECT:2. Verify Close Doors PB command is set for 1000 on Cab1 and General Signal

³ Due to confidentiality agreements, some results points are not displayed.

4.5.Execution and Report

As a trainee I did not carry out this stage, but I have behaved in the realization of reports and necessary documentation for CSW client.

With these reports, it will be possible for the Engineer responsible for the project to review and analyze reported data, thus being able to give approval for the entrance to the same train to enter vehicle testing. Also, in the case of an audit, documents will be required to assess the same.

5. Techniques used in the Project

After having presented the process, in the previous chapter, I will now present the test techniques that were most used in the project.

5.1. Positive/Negative Tests

The Positive / Negative tests represent a perfect analysis of the requirements, they identify the cases that lead to the fulfillment of one of the requirements and the case that do not fulfill the requirements.

Table 12 - Positive/Negative tests

Positive tests	Represent where system works as expected by the requirement.
Negative tests	Represent tests where system does not work

We may have to combine other techniques, since the implementation behavior will not necessarily have the duality "yes or no". Increasing the number of testing also helps, in-depth testing.

5.2. MC/DC

The Modified Condition/Decision Coverage is the most used method, in the project, to converge most possible hypothesis of cases under test, to test most representative cases, to pay attention to the most restrictive conditions. MC/DC requires that each condition in a decision has been shown to independently affect that decision's outcome. Which outcome of a decision changes because of changing a single condition (Rapita Systems, n.d.) (Sá, 2020)

To ensure the test coverage using the MC/DC technique, the following conditions must be complied: (Sá, 2020) (Rapita Systems, n.d.)

- Every entry and exit point are invoked.
- Every decision takes every possible outcome.
- Every condition in a decision takes every possible outcome.
- Every condition in a decision is shown to independently affect the outcome of the decision.

A minimum of $N+1$ action, where N is the number of conditions, can generally be achieved, although some combinations of conditions are redundant.

Example for the requirements above:

Legend:

0 – False

1 – True

X – Unknown

Table 13 - Example of MC/DC.

A (Given)	B (When)	Result (Then)
0	0	X
0	1	X
1	0	X
1	1	1

5.2.1. Used in process.

MC / DC was one of the methods most used by me in the project, since most of the signals I worked were Boolean signals (which have two stages). This methodology considers the combination of the more restrictive cases.

In other cases, it also used MC / DC methodology, being that it approximates Boolean cases, thus considering pressure (for example the pressure of the pipeline) as True and False, for example. Thus, managing to make more efficient TCD.

5.3. Boundary Value Analysis

Boundary value analysis is a technique that comes from Black Box testing (previously referred). Boundary testing is the process of testing for the following values: 0, Default, Minimum, Maximum, nearest adjacent value lower than the minimum and nearest adjacent value. The minimum possible increment should be used. (Guru99_A, n.d.) (Sá, 2020)



Figure 59 - Schematic in example of BVA. (Sharma, 2019)

For this example, it is necessary to perform the following tests:

Valid Inputs: Between 16 and 60

Invalid Inputs: Lower 16 or larger 60

Valid Class: 16 – 60

Invalid Class 1: <16 – Choose values at the border of 16.

Invalid Class 2: >60 - Choose values at the border of 60.

Assuming that the default value is 50, comes:

Table 14 - Example of BVA.

Boundary values or Test	0		16			Default (50)	60		
Action	0	1	15	16	17	50	59	60	61
Results	Invalid	Invalid	Invalid	Valid	Valid	Valid	Valid	Valid	Invalid

In Boundary analysis the most important are the test values in the border.

5.3.1. Used in process.

Following the test idea where it involves pressure, there will be an analysis of the frontier values, pressure to see if it considers what the requirement says, if invalid value is invalid.

5.4. Equivalence Class Partitioning

Equivalence Class Partitioning or Equivalence Partitioning is a type of Black Box testing, which is used for example for unit testing. In this technique, input data units are divided into equivalent partitions that can be used to derive actions. (Guru99_A, n.d.) (madhurihammad, 2020)

Example:

Assuming we must test a field, which accepts 10 – 15 (Example based in (Rajkumar, 2018))

Valid Input: 10 – 15

Invalid Input: less than or equal to 10 (≤ 10), greater than or equal to 15 (≥ 15)

Valid Class: 10 – 15 - Choose any input test data from 10 – 15.

Invalid Class 1: ≤ 10 - Choose any input test data less than or equal to 10.

Invalid Class 2: ≥ 15 - Choose any input test data greater than or equal to 15.

Table 15 - Equivalence Partitioning

Equivalence Partitions		
Invalid	Valid	Invalid
≤ 10	10-15	≥ 15

5.4.1. Used in process.

As the test indicates, the concern of this type of tests is the verification of value ranges, whether they are valid or invalid. We returned to the question of pressure of the pipeline, it may be an indication if pressure found admissible values.

In some cases, it will be relevant to add Boundary analysis to Equivalence Class Partitioning. This analysis is especially relevant when we have systems where the transition of values is important.

6. Problems and Solutions that appeared in the Project

In this chapter I will focus on the problems that were presented during the process of creating tests.

6.1. What problems appear?

During the process that I talked about earlier, there are problems that we will have to solve. These problems can be related to the implementation made by the developers or other problems that come with the simulator that is used.

In the next table I will present some of the principal problems:

Table 16 - Principal problems in Process.

Root of Problem	Description of Problem	Resolution/Type of Problem
Real-time system simulation	State simulates real time system, we may have tuning problems.	Use admissible tolerances
Requirements with problems	Requirements with problems in interpretation.	Open defect for new information
Missing Signals in ISL	There are documented signals necessary for TS	<ul style="list-style-type: none"> • Find signals with another name, use same sign. • Badly implemented requirement (Defect)
Missing Implementation	Requirement not yet implemented	Wait for an upcoming release.

6.1.1. Real-time system simulation

- Admissible tolerances

Due to some delays that are caused, we will try to simulate a real time system. We will have delays and therefore we have to create admissible tolerances to take out the influence of these errors.

For example, for the case presented in the TCD chapter of “Workflow in CSW Project”, for threshold and for pulse, it was necessary to define their tolerances. In this case, they were the following:

```
if (threshold == 2000)
    thresholdsim = threshold * 0.2;
if (pulse == 1000)
    pulsesim = pulse * 0.2;
if (threshold == 20000)
    thresholdsim = threshold * 0.8;
if (pulse == 10000)
    pulsesim = pulse * 0.8;
```

- Simulator runs faster than expected

It may seem strange, after knowing that the system has delays, but there are cases when results appear early than expected, but that may be affected by other requirements.

For example, for the test mentioned above, we have this problem. For Action 1, we expected to have results at the end of 86ms, however a CCUS clock cycle didn't arrive, therefore signals of the combinations never went low, that made the push button switch enter the requirement for degrade mode (so is stayed in degrade mode), and soon it was no longer possible to test, and thus this test failed.

The solution was to wait 1s to check if the signal was reset, this way the combination was disabled at that time.

<p>1. Verify Close Doors PB command is reset on both cabs and General Signal for 1000ms</p>
--

6.1.2. Requirements with problems

Sometimes the requirement may have ambiguities, and it can be difficult to understand, therefore you should never make personal interpretations or considerations about the requirement, and sometimes it is necessary to create a defect so as to ask for clarifications.

For example, I went through a case where we had a requirement to set, but there was no reset requirement. So what was implemented in code was "and" (when one of the conditions was disabled, the reset happened), but I could not interpret that way. So, I had to open a defect to clarify.

6.1.3. Missing Signals in ISL

This happens because some documents are not updated in time. So, we may have outdated information on signal list. Sometimes the signal presented in code may differ from the one in the ISL, in this case we must report this problem to the code developer to update the ISL accordingly.

6.2.Debug of Problems

6.2.1. From TS

From the Script you can debug problems, and you can use the "Wait" function, which makes the CCU to continue with the forced signals and we can observe the state during the "wait" time.

From the Client's own development tool, you can see the code (Function Block Diagram), see the status of the signals in real time.

Using Microsoft Visual debugging will not be useful, as this will stop the CCU simulator. Advantage is lost using this method.

6.2.2. CSW Client IDE

With the client's IDE it is possible to observe code created by the developers, thus being able to find some more visible defects. It is also possible to use CCU simulator, to be able to directly debug code and view code.

6.2.3. Observe wave behavior.

There is also the possibility to observe signal waves over time, as can be seen in the image below.

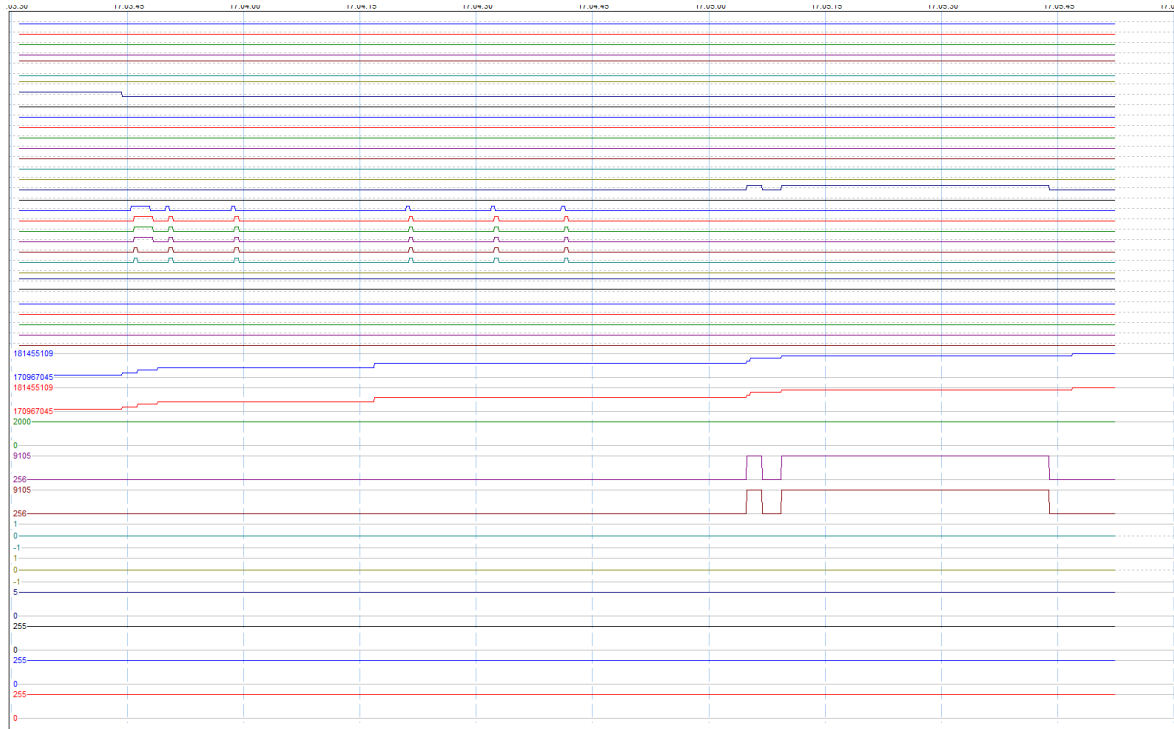


Figure 60 - Observe waves.

7. Automation Tool - Jenkins

To perform automatic tests, a Jenkins was used.

7.1.Jenkins

Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. (Jenkins_A, n.d.)

Jenkins can watch for any code changes in repositories (example of Git) and automatically do a build with tools like Maven and Gradle. We can use container technologies, initiate tests, and then take actions like rolling back or rolling forward in production. (CloudBees, n.d.)

7.2.Jenkins History

The Jenkins project started in 2004 by Kohsuke Kawaguchi (former employee of Sun Microsystems), and Jenkins was originally called Hudson. Kohsuke was a developer, who got tired of incurring the wrath of his team every time his code broke the build. To solve the problem, he created Jenkins to perform continuous integration. Then Kohsuke open sourced it, creating the Jenkins project, and now Jenkins is one of the most used automatic tools. (CloudBees, n.d.)

7.3.Jenkins Pipeline

Pipeline of Jenkins is based on continuous delivery, script automated expression of our process for getting software from version control right through to our users and customers.

An advantage is that every change in software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as the progression of the built software (called a “build”) through multiple stages of testing and deployment. (Jenkins_C)

7.3.1. Jenkins File

Creating a Jenkins file provides several immediate benefits (Jenkins_C):

- Automatically create Pipelines for all Branches and Pull Requests.
- Code review/iteration on the Pipeline.
- Audit trail for the Pipeline.
- Single source of truth (is the practice of structuring information models and associated data schema such that every data element is mastered (or edited) in only one place) for the Pipeline, which can be viewed and edited by multiple members of the project.

```
// Declarative //
pipeline {
    agent any ①

    stages {
        stage('Build') { ②
            steps { ③
                sh 'make' ④
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml' ⑤
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}

// Script //
node {
    stage('Build') {
        sh 'make'
    }

    stage('Test') {
        sh 'make check'
        junit 'reports/**/*.xml'
    }

    stage('Deploy') {
        sh 'make publish'
    }
}
```

Figure 61 - Example of Jenkins file. (*Jenkins_C*)

Legend:

- **pipeline** - Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.
- **agent** - indicates that Jenkins should allocate an executor and workspace for this part of the Pipeline. (Presented in Figure 61 with the number 1 marked in red)
- **stage** - describes a stage of this Pipeline. (Presented in Figure 61 with the number 2 marked in red)
- **steps** - describes the steps to be run in this stage. (Present in Figure 61 with the number 3 marked in red)
- **sh** - executes the given shell command. (Present in Figure 61 with the number 4 marked in red)
- **junit** - Pipeline step provided by the plugin for aggregating test reports. (Present in Figure 61 with the number 5 marked in red)
- **node** - Scripted Pipeline-specific syntax that instructs Jenkins to execute this Pipeline (and any stages contained within it), on any available agent/node. This is effectively equivalent to agent in Declarative Pipeline-specific syntax.

There are two parts of the Script:

- **Declarative:** a relatively recent addition to Jenkins Pipeline which presents a more simplified and opinionated syntax on top of the Pipeline sub-systems. Declarative limits what is available to the user with a more strict and pre-defined structure, making it an ideal choice for simpler continuous delivery pipelines.
- **Scripted:** provides very few limits, insofar that the only limits on structure and syntax tend to be defined by Groovy itself, rather than any Pipeline-specific systems, making it an ideal choice for power-users and those with more complex requirements.

7.3.2. Pipeline Syntax

Jenkins can be scripted in two ways, both from a programming language (Groovy) and in a more simplified way, Blue Ocean, that is more limited because it is not a programming language, which we cannot do some tasks because the pipeline continues, and we do not use cycles (with example for and while).

- Groovy

Apache Groovy is a powerful, optionally typed, and dynamic language, with static-typing and static compilation capabilities, for the Java platform. It integrates smoothly with any Java program, and immediately delivers to your application powerful features, including scripting capabilities, Domain-Specific Language authoring, runtime and compile-time meta-programming and functional programming. (Apache Groovy project, n.d.) In this case that is already present in the Figure 61 .

- Blue Ocean

Blue Ocean is a new user experience for Jenkins based on a personalize, modern design that allows users to graphically create. (Jenkins_B, n.d.)

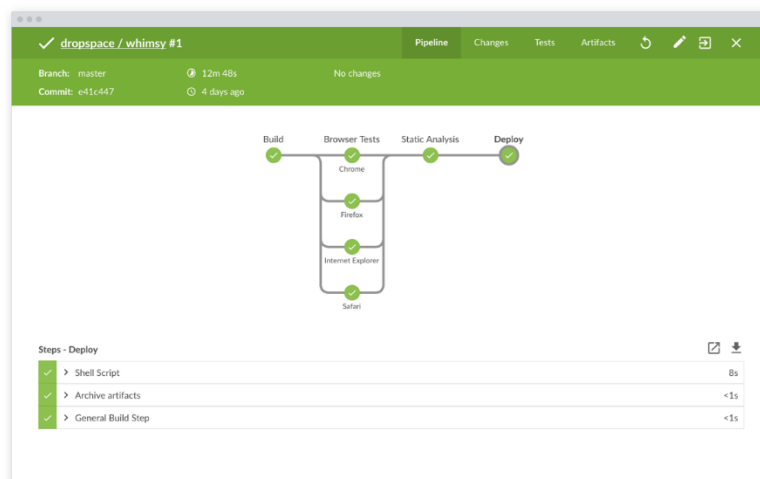


Figure 62 - Example of Blue Ocean (*Jenkins_B, n.d.*)

As can be seen in Figure 62, we always have a simple language, however it becomes limited, so it will not be a good option for something more complex.

7.3.3. Advantages

The advantages in Jenkins process are (Jenkins_C):

- **Code:** Pipelines are implemented in code and typically checked into source control, giving the ability to edit, review (important to be successful in Developers working together), and iterate upon their delivery pipeline.
- **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins master.

- **Pausable:** Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile:** Pipelines support complex real-world continuous delivery requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible:** The Pipeline plugin supports custom extensions to its Domain-Specific Language (computer language specialized to a particular application domain) and multiple options for integration with other plugins.

Example of Jenkins flowchart:

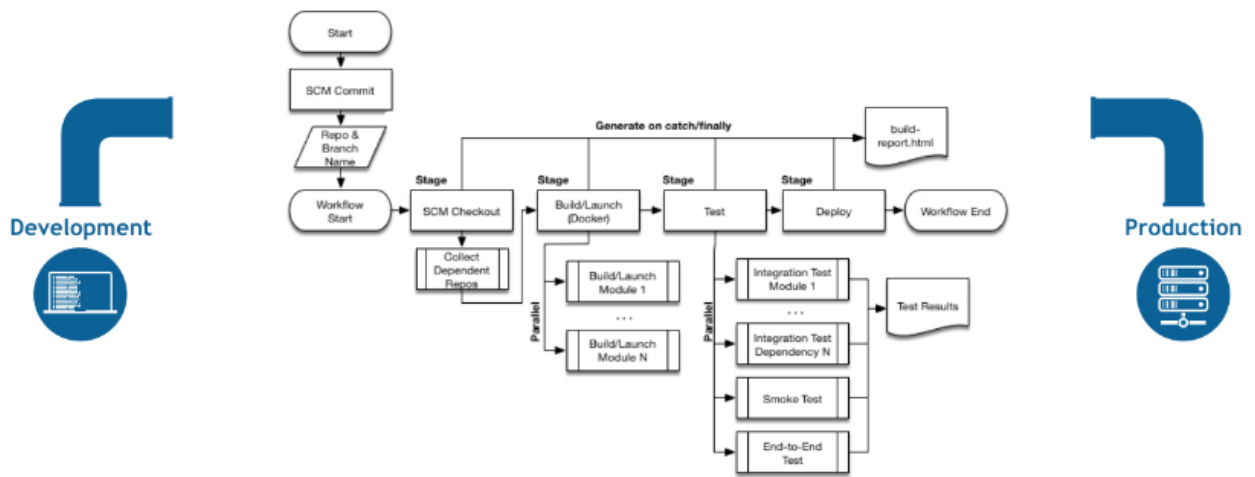


Figure 63 - Pipeline Flow. (Jenkins_C)

7.3.4. Jenkins Workflow

In the Figure 64 we can observe the work done by the developer.

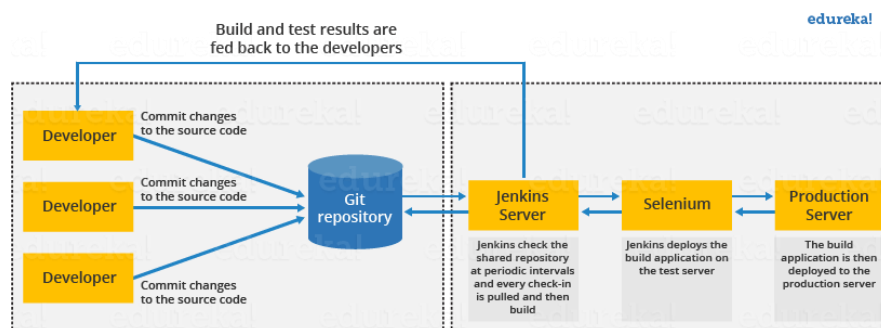


Figure 64 - Flow diagram of Continuous Integration with Jenkins. (Saurabh, 2020)

Description of Figure 64 (Saurabh, 2020):

- First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository, for example looking for changes in regular intervals.
- Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If built is successful, then Jenkins deploys the built in the test server.
- After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.

We have already seen the advantages of having Jenkins in the project, so I will highlight the main ideas:

Table 17 - Non-use and Use of Jenkins. (Saurabh, 2020)

	Non-use	Use
General	The entire source code was built and then tested.	Every commit made in the source code is built and tested.
Testing Results	Developers must wait for test results	Developers know the test result of every commit made in the source code on the run.
Delivery of code	The whole process is manual	You only need to commit changes to the source code and Jenkins will automate the rest of the process for you.
Locating Bugs	Manual Process	Automatic Process
Time Consuming	More difficult and time-consuming,	Automatic Process, developers only need to focus on a particular commit.
New Software	Slows the software delivery process.	Frequent new software releases.

7.4.Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol. (JavaTpoint, n.d.)

Jenkins architecture has two components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server

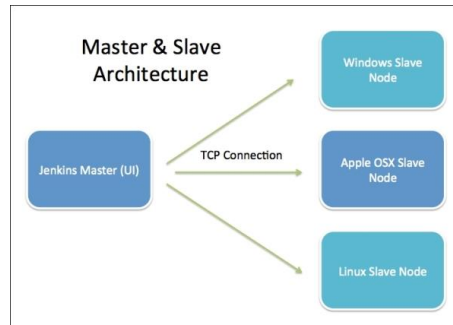


Figure 65 - Jenkins Architecture. (JavaTpoint, n.d.)

7.4.1. Jenkins Master

Master's job is to handle (JavaTpoint, n.d.):

- Scheduling build jobs.
- Dispatching builds to the nodes/slaves for the actual execution.
- Monitor the nodes/slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master/Server instance of Jenkins can also execute build jobs directly.

7.4.2. Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master.

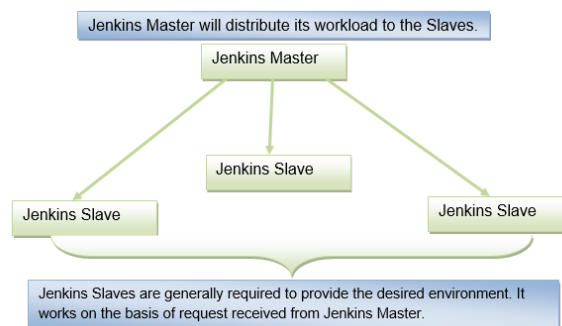


Figure 66 - Jenkins internal Workflow. (JavaTpoint, n.d.)

7.5.Jenkins Multibranch Pipeline

7.5.1. What is it?

The Multibranch Pipeline enables different branches to make several builds (one build for each Branch). In a Multibranch Pipeline project, Jenkins automatically discovers, manages, and executes Pipelines for branches which contain a Jenkins file in source control. (Jenkins, n.d.)

The screenshot shows the Jenkins web interface for a project named 'freestyle-multi-branch'. The left sidebar contains navigation links: Up, Status, Configure, Branch Indexing, Delete Project, People, Build History, Edit View, Move, Job Config History, and Credentials. The main area displays a table of build results for various branches. Below the table are sections for 'Build Queue' (showing no builds) and 'Build Executor Status' (showing one build in progress for the 'development' branch). A legend at the bottom provides links for RSS feeds for all builds, failures, and the latest builds.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		development	36 sec - #3	N/A	31 sec
		feature/cool-stuff	1 min 10 sec - #2	N/A	30 sec
		master	3 min 22 sec - #11	N/A	31 sec
		stable	39 sec - #2	N/A	30 sec

Build Queue: No builds in the queue.

Build Executor Status:

- freestyle-multi-branch » [development](#) [#4](#)
- Idle

Legend: [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Figure 67 - Multibranch Pipeline example. (Jenkins_D, n.d.)

7.6.Jenkins in Project

7.6.1. Why is Jenkins use in the project?

Jenkins was applied to project, to facilitate our process. Jenkins was already used in the project, and his advantage of the analysis level, with the objective of always having updated test logs, to have sequence of the test results in the different builds and corrections that are made, or even new tests.

As this will also lead to the repetition of the tests, we will have a better sampling of the results. So, we are sure of the results, thus creating better results reliability, which will take tests with better quality and robustness. Because we do not want a test pass for yourself, we know that testing is expected behavior.

Table 18 - Advantages and disadvantages Jenkins in the project.

Advantages	Disadvantages
Have updated logs: <ul style="list-style-type: none"> • New Test Case • For new code build 	Long compile and run time
Have resisted previous builds	Have virtually every project run for each build
Reduce time spent running tests	
Data always available	
Find compilation problems	
Creation of robust and quality tests	

Jenkins with this situation presented, comes to facilitate the process. But as I said is time consuming (about two days running all TS). To discover a problem, it will probably be too late. That is why it is necessary to create a process that will discover an error sooner, as quickly as possible.

So, it is necessary to create a process that could analyze this as soon as possible, best way to do analyze branch of new changes. One of these solutions will be to use multibranch.

7.6.2. Why use Multibranch

With Multibranch we can reduce the time of knowing if something is wrong. Without this functionality we would have to wait for the building of the total project (with and without changes in project).

Some of these problems can lead to a pipeline error, thus delaying obtaining the logs. Some of these problems are:

- Poorly performed functions.
- Undeclared signals in the project.
- Problems of synthase in TS.

With this delay we will not have logs in time and that leads to delays. To solve this situation, we can analyze branch by branch, to find the branch who was a new code, and then compile a new code, for example.

7.6.3. Work Done

By the analysis of the process that was developed previously, despite already presenting an analysis of the Pull Request (PR), we came to the point where we ask if something else could be used to improve it, and therefore Multibranch was applied.

In this way, we were able to improve the process, namely, to find out if a new code for compiled.

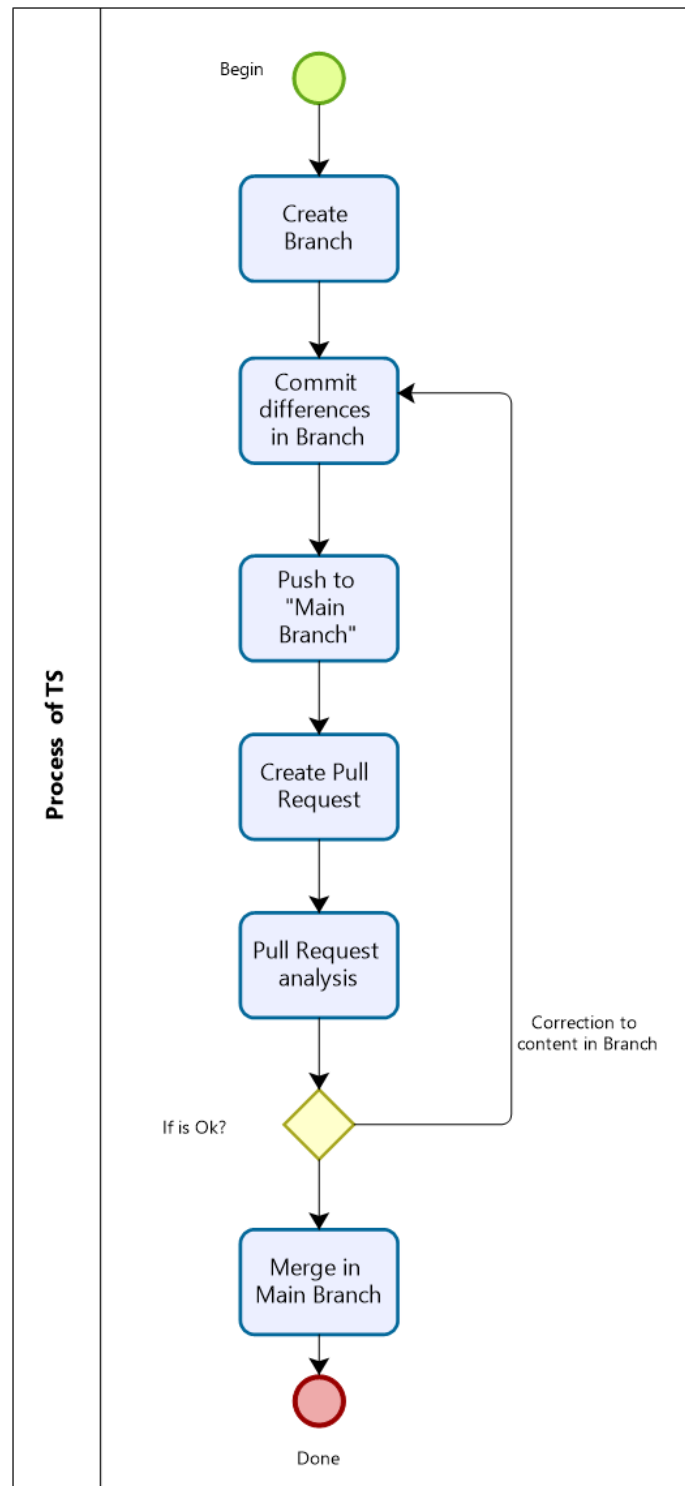


Figure 68 - Process of TS.

Some topics to show that it was done with Multibranch:

- **Identification from Branch** name of what you want to analyze, with Regex feature.
 - According to the project process, branch names must respect rules to be created. Thus, it is possible to identifying necessary data, for build only necessary, so regex was used to identify data from the name of the branch.
- **Checkout repository** in Branch in analysis.
 - To build the project, we will need files that exist in repository. So, it will be done a checkout branch which we are used. For this, Jenkins native git functions are used for this function, which will store files in the Jenkins build workspace.
- **Pre-Build**, checking the type of test that needs to be done, location of some files.
 - Files from the software simulator are in a zip folder, so you need to unzip them. It is also necessary to make changes to a batch file that do in this step.
 - Filling in the generic tool configuration file. For that, Python script was used to fill that same file. So, with the ElementTree package, it is identified parameters of configuration and at the same time fill them.
- **Build Project**, only the one in analysis (note that in the repository there is more than one build, so select the Branch name that matters)
 - As previously indicated, we used C # with Microsoft visual. So, to build a project, it is used a MS Build.
- **Clean WORKSPACE**, Stage where to delete Jenkins workspace, so as not to leave files in computer that will no longer be needed, also in case of failure there may be a waiting period time for the user to observe, what's wrong in workspace, have possibility even not letting files be deleted.

The following figure shows the flowchart of the pipeline:

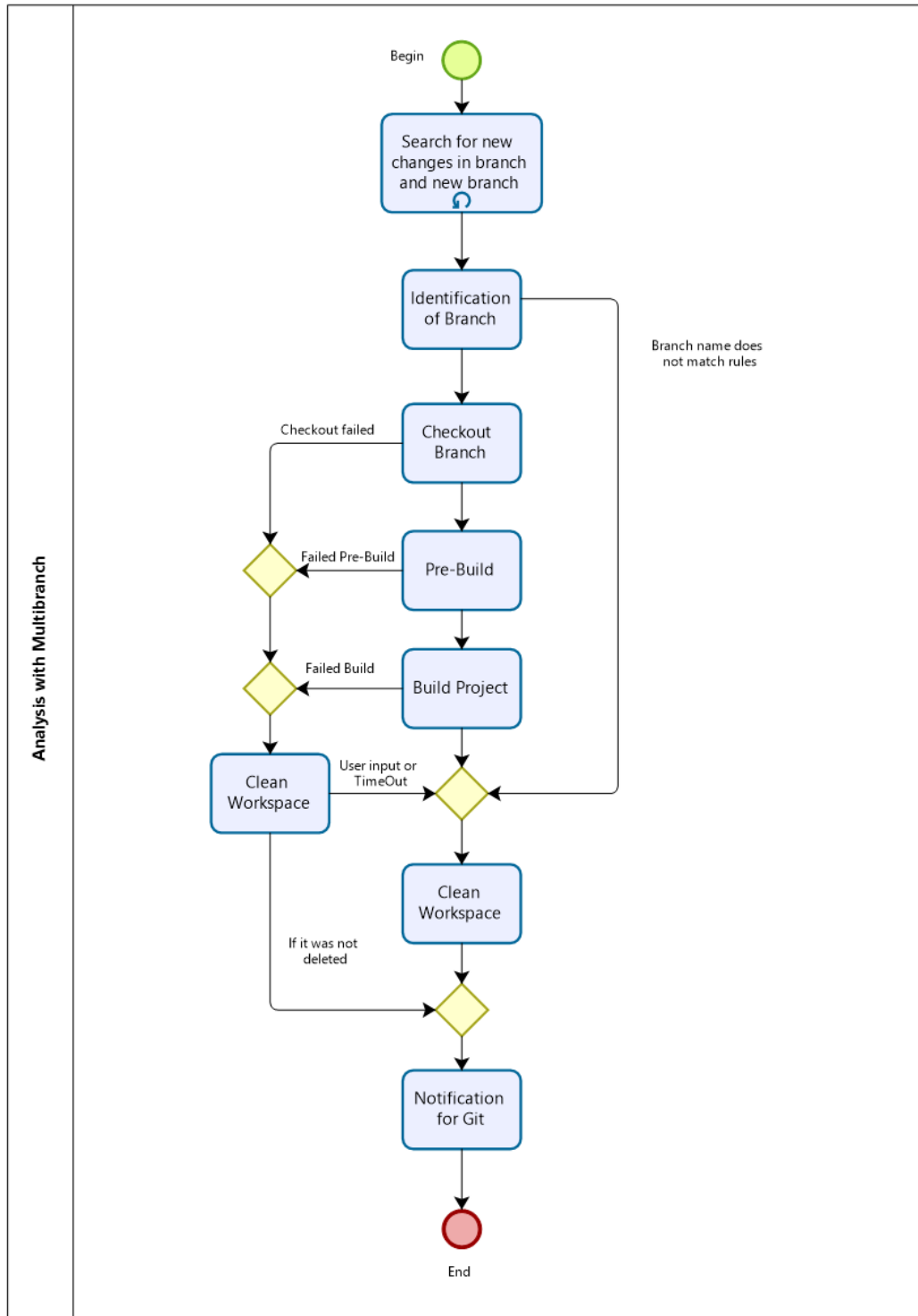


Figure 69 - Multibranch process.

In the end we will observe an environment identical to the one in the Figure 67.

Below are presented some parts of the Jenkins File⁴:

- Build Version:

Definition of pipeline input parameters. In this case "build number" is present. If the build is running when changed, the same parameter is disabled.

```
properties([
    disableConcurrentBuilds(),
    parameters([
        string(name: 'BUILD_NUMBER', defaultValue: '//(NumBer)'),
    ])
])
```

- Branch Name Verifications:

Branch check to verify that it is within the parameters and that we are testing safe or not safe code.

```
stage('Check branch name') {
    BRANCH_PATTERN = ~ //(Branch_Regex)

    def matcher = (BRANCH_NAME =~ BRANCH_PATTERN)
    if(matcher){
        //Debug vars
        //Confidential

        ccuo = matcher.group("ccuo")
        ccus = matcher.group("ccus")
        bb = ccuo ? ccuo : ccus

    }else{
        errArray << 1
        error "Branch does not follow the template"
    }
}
```

⁴ In annex, I will present Jenkins File, due to confidentiality agreements, I will only present it on paper in the presentation day. In Appendix B I present Jenkins file - complete file with some erasures.

- Checkout of Branch:

Checkout of the branch under test.

```
stage('Checkout repository') {
    echo '[INFO] Cloning repository'
    checkout([$class: 'GitSCM', branches: [[name: BRANCH_NAME]], doGenerateSubmoduleConfigurations: false, extensions: [[$class: 'CleanBeforeCheckout']], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '(/ID)', url: '(/GIT_REPOSITORY_URL)']]])
}
```

- Name of Paths and Unzip of Products:

Pre-build, zip build of the code under test and changes necessary for your test function.

```
def buildName = "(/ProjectDATA))_${params.BUILD_NUMBER}"
currentBuild.description = "(/ProjectDATA))_${params.BUILD_NUMBER}"

stage('Pre-Build') {
    def buildsFolder = "./builds/${buildName}"
    def buildZip = ccuo != null ? "(/CCUO_BuildFile)" : "(/CCUO_BuildFile)"

    if( fileExists("${buildsFolder}/${buildZip}") ){
        //Unzipping software build
        dir(buildsFolder){
            try {
                def unzipped = unzip zipFile: buildZip, quiet: true

                //Replace text in file
                if(ccuo) {
                    //(/Confidential)
                } else {
                    //(/Confidential)
                }
            } catch(Exception e){
                errArray << 2
                error "Failed to unzip test build"
            }
        }
    } ...
}
```

- Build of Project:
Build the project with MSBuild (compiler).

```
stage('Build Projects') {
    dir("./${directory}") {
        try{
            powershell "${CSharpCompiler} Testcase.sln -
p:Configuration=Release"
        }catch(exception){
            errArray << 4
            error "Failed to compile ${directory}"
        }
    }
}
```

- Clear workspace:
Clean Workspace of the branch under test.

```
if(shouldDeleteWS){
    stage('Clean WORKSPACE'){
        powershell 'Get-ChildItem . -Force | Remove-Item -Recurse -
Force'
        dir("${WORKSPACE}@script")
        {
            powershell 'Get-ChildItem . -Force | Remove-Item -
Recurse -Force'
        }
    }
}
```

7.6.4. Possible improvements

For this improvement, the use of a virtualization tool (Docker, for example) was thought to create virtual environments to be able to run between different tests at the same time (because a simulator cannot be performed in this case more than one test at the same time).

Then the container would be used to make a simulation in each Test Case, but for that it will be necessary to carry out the following:

- Creating a Windows Container.
- Installation of CSW Client test tools in container.

Then Jenkins integration with virtualization tool to run more sequence tests for each branch.

8. Programs Used in the Project

In this chapter I will describe some programs. Some of them were unknown to me and I needed to do some research, and there were others that I already knew and gained new skills.

8.1. Languages used.

8.1.1. C#

Throughout the project I used C # to implement TCDs previously made.⁵ Briefly, C# is a simple, modern, object-oriented, and type-safe programming language. This is an object-oriented language, but C# further includes support for component-oriented programming. C# provides language constructs to directly support these concepts, making C# a very natural language in which to create and use software components. (Microsoft, 2017)

Some of C# features are: (Microsoft, 2017):

- Garbage collection automatically reclaims memory occupied by unused objects.
- Exception handling provides a structured and extensible approach to error detection and recovery.
- Type-safe design of the language makes it impossible to read from uninitialized variables to index arrays beyond their bounds, or to perform unchecked type casts.

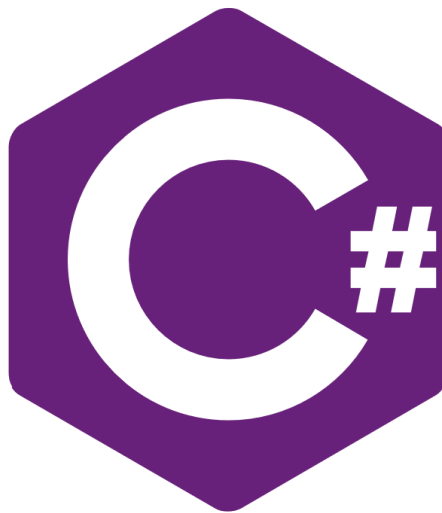


Figure 70 - C# Logo. (Techbaz, n.d.)

⁵ Chapter 4 presented excerpts from TS as an example of what was made.

8.1.2. Groovy

Throughout the project I used Groovy to implement Jenkins File to create the multibranch.⁶ Briefly, the Apache Groovy is a Java-syntax-compatible object-oriented programming language for the Java platform. It can be used both as a programming language and a scripting language for the Java Platform, it is compiled to Java virtual machine (JVM) bytecode and interoperates seamlessly with other Java code and libraries. Groovy uses a curly-bracket syntax like Java's. Groovy supports closures, multiline strings, and expressions embedded in strings. (Wikipedia_F, 2021) (Apache Groovy, n.d.)



Figure 71 - Groovy Logo. (Wikipedia_F, 2021)

Some of Groovy's features are:



Flat learning curve

Concise, readable and expressive syntax, easy to learn for Java developers



Powerful features

Closures, builders, runtime & compile-time meta-programming, functional programming, type inference, and static compilation



Smooth Java integration

Seamlessly and transparently integrates and interoperates with Java and any third-party libraries



Domain-Specific Languages

Flexible & malleable syntax, advanced integration & customization mechanisms, to integrate readable business rules in your applications



Vibrant and rich ecosystem

Web development, reactive applications, concurrency / asynchronous / parallelism library, test frameworks, build tools, code analysis, GUI building



Scripting and testing glue

Great for writing concise and maintainable tests, and for all your build and automation tasks

Figure 72 - List of resumes of vantages. (Apache Groovy, n.d.)

⁶ Chapter 5 presented excerpts from Jenkins file as an example of what was made.

8.1.3. Python

Throughout the project I used Python to interpreter some scrips that were done and used by the team. One of the points where it was important to use python to process the Simulator configuration file. An XML file is required, where data is present. For that we used ElementTree's package to fill in data.

Briefly, the Python is an interpreted, high-level, and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers to write clear, logical code for small and large-scale projects. (Kuhlman, 2012) (Wikipedia_K, 2021)

Some of Python ideas are the following:

- Code is automatically compiled to byte code and executed, Python is suitable for use as a scripting language, Web application implementation language, etc.
- Python can be extended in C and C++, Python can provide the speed needed for even compute intensive tasks.
- Because of its strong structuring constructs (nested code blocks, functions, classes, modules, and packages) and its consistent use of objects and object-oriented programming, Python enables us to write clear, logical applications for small and large tasks.



Figure 73 - Python Logo. (Python, n.d.)

Some of Python features are (Kuhlman, 2012):

- Built-in high level data types: strings, lists, dictionaries, etc.
- The usual control structures: if, if-else, if-elif-else, while, plus a powerful collection iterator (for).

- Multiple levels of organizational structure: functions, classes, modules, and packages. These assist in organizing code. An excellent and large example is the Python standard library.
- Compile on the fly to byte code - Source code is compiled to byte code without a separate compile step. Source code modules can also be "pre-compiled" to byte code files.
- Object-oriented - Python provides a consistent way to use objects: everything is an object. And, in Python it is easy to implement new object types (called classes in object-oriented programming).
- Extensions in C and C++ - Extension modules and extension types can be written by hand. There are also tools that help with this, for example, SWIG, sip, Pyrex.
- Jython is a version of Python that "plays well with" Java.

8.2.GIT

I will cover some concepts of Git, which is associated with Continuous Integration. The project is also used to review the TCD and TS (code reviews), thus having a track of changes that have been made, while having a track that did what and who did the same review. So also, from there that Jenkins will also track the situation changes in the branches themselves.

8.2.1. What is Git?

Git is a distributed version control system (system that records changes to a file, or set of files, over time so that we can recall specific versions later). It keeps track of projects and files as they change over time with the help of different developers. (Edpresso Team, n.d.)

Git helps keep track of changes made to a code. If we have problems in middle of the process, Git allows you to revert code to stable state. It also helps to make track changes in the code that was already made.

In the project where it was inserted, Git was used to follow up on changes, to make code reviews, to new tests. Jenkins's associate can build the code and verify it.

Git was originally created by Linus Torvalds in 2005 (Wikipedia_I, 2020).

8.2.2. How Git works

Here is a basic overview of how Git works (Atlassian, n.d.):

1. Create a "repository" (project) with a Git hosting tool.
2. Clone the repository to your local machine.
3. Add a file to your local repo and "commit" the changes.
4. "Push" your changes to your master branch.
5. Make a change to your file with a git hosting tool and commit.
6. "Pull" the changes to your local machine.
7. Create a "branch" (version), make a change, commit the change.
8. Open a "pull request" (propose changes to the master branch).
9. "Merge" your branch to the master branch.

8.2.3. Glossary

Table 19 - Glossary of GIT.

Repository	A repository is a collection of source code. A repository has commits to the project or a set of references to the commits.
Commits	A commit logs a change or series of changes that you have made to a file in the repository.
Branches	A branch is essentially a unique set of code changes with a unique name. Each repository can have one or more branches. <ul style="list-style-type: none"> • Main Branch (can be Master, for example) — the branch where all the changes eventually get merged into.
Fork	A fork is a copy of a repository. Forking a repository allows you to experiment at will without compromising the original design.
Pull Request	Pull requests serves to give identification of files that have been changed. Once a pull

	request is opened, can discuss, and review the potential changes, add follow-up commits before your changes are merged into for example master branch. (Github, n.d.)
--	---

8.3. MS Build

MS Build to build .NET and Visual Studio projects. It was thought to use a MS Build as compiler C#.

- **What is MS Build?**

The Microsoft Build Engine is a platform for building applications. This engine, which is also known as MS Build, provides an XML schema for a project file that controls how the build platform processes and builds software. (Microsoft, 2016)

8.4. CSW client Tools

Here I will present some of the tools used⁷:

- Tool testing in C#: This was the tool most used by me, as in the tests that were carried out. The tool as I mentioned earlier is a C # based tool.
- Debug Tool: This tool is a code debugging tool where it is possible to see running code (i.e., Function Block Diagram) in real time, thus analyzing possible failures.
- Signal Analyzer: This tool is used to be able to analyze the signals that are running CCU.

⁷ Due to confidentiality agreements, I cannot get into detail, it is the idea of functionality.

9. Other Activities

In this chapter I will address, training programs that I attended during the internship in CSW.

9.1. Newcomers Training

In the first training, I was introduced to the company, the values, the way it works, and the software used.

In short, those were the first steps to start in the company.

9.2. “Internal Training”

With this training the railway world was introduced to me. Tools and knowledge necessary to integrate the project and team, were given me. Some of these issues of this training already discussed in this dissertation and consequently were deepened with acquired knowledge. Therefore, in summary the subjects that were presented to me in this training were:

- Railway Components.
- Systems in Train.
- TCMS architecture and its complexity.
- Development environment.
- Development standards safety-critical.
- Security for industrial automation and control systems.
- Information security management.
- Among other topics.

9.3. Basic Training

This training was divided into two parts, “Basic Training - Processes & Tools” and “Basic Training - Software Development Process”. This training’s goal was to introduce some concepts and tools used in CSW. I will present below those two parts.

9.3.1. Basic Training - Processes & Tools⁸

In this training, that was more focused on the company's internal knowledge and on the work developed by the company. This training provided me with some software life cycle knowledge and an introduction to Git tool, among other tools.

Some subjects covered in this training were:

- Project Management and Life Cycle.
- Development methodologies.
- Support activities.
- Support tools.
- Pull-request Exercise.
- Innovation & Knowledge.

9.3.2. Basic Training - Software Development Process⁹

In this training, that was more focused on practical knowledge, the trainees had to go through a practical exercise to demonstrate the software life cycle.

Some subjects covered in this training were:

- Software Development Process (SDP) Overview.
- Enterprise Architecture (EA) introduction and CSW template.
- Requirements Analysis – Concepts & Processes.
- Requirements with EA.
- Software Design - Concepts & Processes.
- Unified Modeling Language (UML) & UML with EA.
- Software Construction - Concepts & Processes.
- Software Testing - Concepts & Processes.
- Hands-On Project with Scrum ceremonies.

⁸ In appendix A, my diploma from this training will be presented.

⁹ In appendix A, my diploma from this training will be presented.

10. Conclusions

The opportunity to carry out this curricular internship at Critical Software was an opportunity for professional development, but also for personal development, as it enabled me a glance of business and work environment. This internship made it possible for me to make a first contact with the world of work, in the Railway area, allowing a very positive contribution to my future, given the set of skills and knowledge that were provided.

The introduction of Railway in the initial training gave me the first skills in this work field. In this training, I was involved in a more generalist training that gave me the general concepts and an introduction, and later the project gave me a deeper view and knowledge of the concept.

Then I had contact with tests, something that I was unaware. So, it took me some time to learn and understand some issues. I think that all problematic situations have been successfully overcome, with time.

Automation tool showed me a new environment that presented some complexity but with time and some support my goals were achieved, which in the end provided me some joy because of the new knowledge that brought me.

Also, I had a new view on Git and its potential in the business environment, which is very useful, if we could add methodologies of CI and CD to the Git applications, it would make easier the work.

The writing of this report is a process that started from the beginning of my internship, and it helped me to better assimilate and consolidate new concepts and new ideas.

As a conclusion, this internship led me to both technical and personal development since it allowed an experience in the job market and in areas that are not so directly linked to my academic training, as for example the use and knowledge of Automation Tool. On the other hand, my knowledge about trains was not very deep before this internship. All the knowledge acquired will stay with me enriched my skills and expertise.

References

- AHMAD, M. (2020, January 14). *Embedded basics Part 1: IEC 61508 functional safety for MCUs*. Retrieved March 7, 2021, from Microcontroller Tips: <https://www.microcontrollertips.com/embedded-basics-iec-61508-functional-safety-mcus-part-1-faq/>
- Aluminium Manufacture. (n.d.). *Rail vehicle body aluminum deep processing in China*. Retrieved November 4, 2020, from Aluminium Manufacture: <http://www.aluminiummanufacturer.com/blog/rail-vehicle-body-aluminum-deep-processing-in-china/>
- Apache Groovy. (n.d.). *Apache Groovy*. Retrieved February 9, 2021, from Apache Groovy: <https://groovy-lang.org/>
- Apache Groovy project. (n.d.). *Apache Groovy*. Retrieved October 29, 2020, from Apache: <http://groovy-lang.org/>
- ATC Team. (2019, May 29). *What Is Continuous Delivery? The Benefits and Best Practices*. Retrieved October 27, 2020, from DZone: <https://dzone.com/articles/what-is-continuous-delivery-the-benefits-and-best>
- Atlassian. (n.d.). *Getting Git Right*. Retrieved October 28, 2020, from Atlassian: <https://www.atlassian.com/git>
- Bombardier_A. (n.d.). *(Very) high-speed redefined*. Retrieved November 4, 2020, from Bombardier: <https://rail.bombardier.com/en/solutions-and-technologies/mainline/high-speed.html>
- Bombardier_B. (n.d.). *TCMS ENGINEERING*. Retrieved October 26, 2020, from Bombardier: <https://uk.bombardier.com/content/dam/Websites/gb/supporting-documents/Careers/BT-TCMS-Engineering.pdf>
- Bombardier_C. (n.d.). *Meeting the capacity challenge*. Retrieved October 20, 2020, from Bombardier: <https://rail.bombardier.com/en/solutions-and-technologies/signalling-and-infrastructure/communications-based-train-control.html>
- Borges, V. (2017, May 17). *Terminology Explained: What is Safety Integrity Level (SIL)?* Retrieved November 16, 2020, from DNV.GL: <https://blogs.dnvgl.com/software/2017/05/what-is-safety-integrity-level-sil/>
- Brosgol, B., & Comar, C. (2010). DO-178C: A New Standard for Software Safety Certification. *SSTC 2010*. Salt Lake City: AdaCore. Retrieved October 14, 2020, from <https://apps.dtic.mil/dtic/tr/fulltext/u2/a558107.pdf>
- CENELEC. (2011). Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. Brussels, Belgium. Retrieved October 12, 2020, from <https://cdn.standards.iteh.ai/samples/20508/f059ecd1111415dbf91a13af058154c/SIST-EN-50128-2011.pdf>
- CENELEC. (2017, October). Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 1: Generic RAMS Process. Brussels, Belgium. Retrieved October 12, 2020, from https://infostore.saiglobal.com/preview/257708547430.pdf?sku=874745_SAIG_NSAI_NSAI_2079592
- CENELEC. (2018). Railway applications -Communication, signalling and processing systems - Safety related electronic systems for signalling. Brussels, Belgium. Retrieved October 13, 2020, from <https://cdn.standards.iteh.ai/samples/60242/bc1b528c1dc3471ab2efd3863f9d718a/SIST-EN-50129-2019.pdf>
- CloudBees. (n.d.). *What is Jenkins?* Retrieved October 19, 2020, from CloudBees: <https://www.cloudbees.com/jenkins/what-is-jenkins>
- Connor, D. P., & Schmid, P. F. (n.d.). *Train Protection*. Retrieved November 13, 2020, from The Railway Technical: <http://www.railway-technical.com/signalling/train-protection.html>
- CONTINUOUS DELIVERY. (n.d.). *What is Continuous Delivery?* Retrieved October 27, 2020, from CONTINUOUS DELIVERY: <https://continuousdelivery.com/>
- Cordeiro, R., & Nunes, R. (n.d.). *TC Design*. Retrieved April 29, 2021, from Critical Software Internal.
- Cordeiro, R., & Serra, P. (2021, March 24). *Way to Work*. Retrieved April 28, 2021, from Critical Software (Internal).
- Costa, H. (2009). *Desenvolvimento formal de um sistema de sinalização ferroviária de acordo com o normativo CENELEC usando o SCADE*. Universidade da Beira Interior, Departamento de

- Informática, Covilhã. Retrieved October 12, 2020, from <https://ubibliorum.ubi.pt/bitstream/10400.6/3746/1/thesis.pdf>
- CRITICAL Software_A. (n.d.). *Our Culture & history*. Retrieved October 9, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/our-world/culture-history>
- CRITICAL Software_B. (n.d.). *Industries*. Retrieved October 9, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/industries>
- Critical Software_C. (n.d.). *Somos a Critical Software*. Retrieved March 7, 2021, from Critical Software: <https://www.criticalsoftware.com/pt/our-world/cultura-historia>
- CRITICAL Software_D. (n.d.). *Our quality promise*. Retrieved October 9, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/our-world/our-promise>
- CRITICAL Software_E. (n.d.). *Railway*. Retrieved December 21, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/industries/railway>
- CRITICAL Software_F. (n.d.). *Safety-Critical V&V*. Retrieved December 21, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/industries/railway/solution/safety-critical-vv-railway>
- CRITICAL Software_G. (n.d.). *RAMS & Certification Support*. Retrieved December 21, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/industries/railway/solution/rams-certification-support-railway>
- CRITICAL Software_H. (n.d.). *Embedded Software Development*. Retrieved December 21, 2020, from CRITICAL Software: <https://www.criticalsoftware.com/en/industries/railway/solution/embedded-software-development-railway>
- Cross. (n.d.). *Determining Safety Integrity Levels for your Process Application*. Retrieved November 16, 2020, from Cross: <https://www.crossco.com/resources/articles/determining-safety-integrity-levels-for-your-process-application/>
- Dietrich, E. (n.d.). *What is Static Analysis? An Explanation for Everyone*. Retrieved October 16, 2020, from ndepend: <https://blog.ndepend.com/static-analysis-explanation-everyone/>
- Dobra, A. (2019, January 9). *White Box Testing: Basics, Advantages, and More (Tutorial)*. Retrieved April 29, 2021, from QA By Example: <https://qabyexample.com/white-box-testing-tutorial-basics-advantages/>
- Edpresso Team. (n.d.). *What is Git?* Retrieved October 28, 2020, from Educative: <https://www.educative.io/edpresso/what-is-git>
- EV dynamic. (n.d.). *Communication Systems*. Retrieved November 11, 2020, from EV dynamic: <https://ev-dynamic.com/communication-systems/>
- Functional Testing*. (2020, September 7). Retrieved October 14, 2020, from Software Testing Fundamentals: <https://softwaretestingfundamentals.com/functional-testing/>
- Github. (n.d.). *About pull requests*. Retrieved November 25, 2020, from Github: <https://docs.github.com/en/free-pro-team@latest/github/collaborating-with-issues-and-pull-requests/about-pull-requests>
- Guru99_A. (n.d.). *Boundary Value Analysis & Equivalence Partitioning with Examples*. Retrieved November 20, 2020, from Guru99: <https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html>
- Guru99_B. (n.d.). *What is BLACK Box Testing? Techniques, Example & Types*. Retrieved October 15, 2020, from Guru99: <https://www.guru99.com/black-box-testing.html>
- Guru99_C. (n.d.). *What is Functional Testing? Types & Examples (Complete Tutorial)*. Retrieved October 14, 2020, from Guru99: <https://www.guru99.com/functional-testing.html>
- Guru99_D. (n.d.). *What is WHITE Box Testing? Techniques, Example & Types*. Retrieved April 29, 2021, from Guru99: <https://www.guru99.com/white-box-testing.html>
- Hilderman, V. (2014). *DO-178 Introduction Whitepaper*. Retrieved March 7, 2021, from Afuzion: <https://afuzion.com/do-178-introduction/>
- Hitachi Rail. (n.d.). *Comparison between the Japanese and European system*. Retrieved November 11, 2020, from Hitachi Rail: https://www.hitachi-rail.com/products/rolling_stock/tilting/feature05.html
- IBM Cloud Education. (2020, January 6). *Docker*. Retrieved October 29, 2020, from IBM: <https://www.ibm.com/cloud/learn/docker>
- IEC. (n.d.). *Functional Safety*. Retrieved October 12, 2020, from IEC: <https://www.iec.ch/functionalsafety/explained/>
- IEC. (n.d.). *IEC 62443*. Retrieved October 14, 2020, from IEC: <https://syc-se.iec.ch/deliveries/cybersecurity-guidelines/security-standards-and-best-practices/iec-62443/>
- Imperva. (n.d.). *ISO/IEC 27001*. Retrieved October 14, 2020, from Imperva: <https://www.imperva.com/learn/data-security/iso-27001/>

- International Software Testing Qualifications Board. (2019, November 11). *Certified Tester Foundation Level Syllabus*. Retrieved February 15, 2021, from International Software Testing Qualifications Board: <https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html>
- ISO. (n.d.). *ISO/IEC 27001*. Retrieved October 14, 2020, from ISO: <https://www.iso.org/isoiec-27001-information-security.html>
- Jain, M. (2020, August 5). *Differences between Black Box Testing vs White Box Testing*. Retrieved April 29, 2021, from Geeks for Geeks: <https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/>
- JavaTpoint. (n.d.). *Jenkins Tutorial*. Retrieved October 20, 2020, from JavaTpoint: <https://www.javatpoint.com/jenkins>
- Jenkins. (n.d.). *Branches and Pull Requests*. Retrieved November 25, 2020, from Jenkins: <https://www.jenkins.io/doc/book/pipeline/multibranch/>
- Jenkins. (n.d.). *JUnit*. Retrieved October 29, 2020, from Jenkins: <https://plugins.jenkins.io/junit/#documentation>
- Jenkins. (n.d.). *Pipeline*. Retrieved October 19, 2020, from Jenkins: <https://www.jenkins.io/doc/book/pipeline/>
- Jenkins. (n.d.). *TAP*. Retrieved October 29, 2020, from Jenkins: <https://plugins.jenkins.io/tap/>
- Jenkins_A. (n.d.). *Jenkins User Documentation*. Retrieved October 19, 2020, from Jenkins: <https://www.jenkins.io/doc/#jenkins-user-documentation>
- Jenkins_B. (n.d.). *About Blue Ocean*. Retrieved October 29, 2020, from Jenkins: <https://www.jenkins.io/projects/blueocean/about/>
- Jenkins_C. (n.d.). *Jenkins User Handbook*. Retrieved October 21, 2020, from Jenkins: <https://www.jenkins.io/user-handbook.pdf>
- Jenkins_D. (n.d.). *Multi-Branch Project (DEPRECATED)*. Retrieved November 26, 2020, from Jenkins: <https://plugins.jenkins.io/multi-branch-project-plugin/>
- Johnson, P. (2020, November 12). *White Box Testing Guide*. Retrieved May 1, 2021, from White Source: <https://www.whitesourcesoftware.com/resources/blog/white-box-testing>
- Kuhlman, D. (2012, April 22). *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Retrieved February 9, 2021, from The Wayback Machine: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html#introduction-python-101-beginning-python
- Leroy Automation. (n.d.). *TCMS Products*. Retrieved November 4, 2020, from Leroy Automation: <https://www.leroy-automation.com/gammes/tcms/>
- Li, J. (2019). A Glance at Transit System Safety. *International Journal of Mathematical, Engineering and Management Sciences*. Retrieved November 11, 2020, from https://www.researchgate.net/publication/336985336_A_Glance_at_Transit_System_Safety
- LPA Group. (n.d.). *Smart Lighting Control Unit*. Retrieved November 11, 2020, from LPA Group: <https://www.lpa-group.com/product/smart-lighting-control-unit/>
- Luger, C., Kallinovskiy, J., & Rieberer, R. (2016). Identification of representative operating conditions of HVAC systems in passenger rail vehicles based on sampling virtual train trips. *Advanced Engineering Informatics*. Retrieved November 11, 2020, from <https://www.sciencedirect.com/science/article/abs/pii/S1474034616300210>
- madhurihammad. (2020, August 28). *Equivalence Partitioning Method*. Retrieved December 22, 2020, from GeeksforGeeks: <https://www.geeksforgeeks.org/equivalence-class-testing-next-date-problem/?ref=lbp>
- Mannion, P. (2017, June 7). *ISO 26262 compliance is not a costly overhead*. Retrieved December 21, 2020, from u-blox: <https://www.u-blox.com/en/blogs/guest-blogs/iso-26262-compliance-not-costly-overhead>
- Mezquita, T. (2020, March 4). *Static Code Analysis*. Retrieved October 16, 2020, from CyberHoot: <https://cyberhoot.com/cybrary/static-code-analysis/>
- Microsoft. (2016, April 11). *MSBuild*. Retrieved October 29, 2020, from Microsoft: <https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2019>
- Microsoft. (2017, June 1). *Introduction*. Retrieved February 9, 2021, from Microsoft: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/language-specification/introduction>
- Myklebust, T., Stålhane, T., & Hanssen, G. K. (2015). *Important considerations when applying other models than the Waterfall/V-model when developing software according to IEC 61508 or EN 50128*. ISSC.

- Retrieved November 11, 2020, from https://www.researchgate.net/publication/281246322_Important_considerations_when_applying_our_models_than_the_WaterfallV-model_when_developing_software_according_to_IEC_61508_or_EN_50128
- Osma, A. (2018, November 17). *How MC/DC Can Speedup Unit Test Creation*. Retrieved November 20, 2020, from Arjay Osma: <https://medium.com/@arjayosma/how-mc-dc-can-speedup-unit-test-creation-752f0fd9638c>
- PagerDuty. (n.d.). *What is Continuous Integration?* Retrieved December 21, 2020, from PagerDuty: <https://www.pagerduty.com/resources/learn/what-is-continuous-integration/>
- Pittet, S. (n.d.). *Continuous integration vs. continuous delivery vs. continuous deployment*. Retrieved October 27, 2020, from Atlassian: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>
- Python. (n.d.). *Python*. Retrieved February 9, 2021, from Python: <https://www.python.org/>
- QATestLab. (2018, March 14). *The Fundamentals of Black Box Testing*. Retrieved October 15, 2020, from QATestLab: <https://qatestlab.com/resources/knowledge-center/black-box-testing/>
- R, S. (2019, December 24). *Functional Testing – Everything You Need To Know*. Retrieved October 14, 2020, from QA Touch: <https://www.qatouch.com/blog/functional-testing-everything-you-need-to-know/>
- RailSystem. (2015). *Rolling Stock Components*. Retrieved October 19, 2020, from RailSystem: <http://www.railsystem.net/rolling-stock-components/>
- RailSystem. (n.d.). *Balise*. Retrieved October 19, 2020, from RailSystem: <http://www.railsystem.net/balise/>
- RailSystem. (n.d.). *Communications-Based Train Control (CBTC)*. Retrieved October 20, 2020, from RailSystem: <http://www.railsystem.net/communications-based-train-control-cbtc/>
- Railway East. (n.d.). *Passenger Information System (PIS) Solution for Railway Emu/Lrt/Metro/Tram/Coach Train*. Retrieved November 4, 2020, from Focus Technology: <https://eastrailway.en.made-in-china.com/product/BjEmUadTHCVg/China-Passenger-Information-System-PIS-Solution-for-Railway-Emu-Lrt-Metro-Tram-Coach-Train.html>
- Railway Signalling Concepts. (2019, September 4). *Automatic Train Protection Railway Signalling Equipment*. Retrieved November 13, 2020, from Railway Signalling Concepts: <https://www.railwaysignallingconcepts.in/automatic-train-protection-railway-signalling-equipment/>
- Railway-News. (2018, June 19). *BT Cables On Track for Significant Business Growth*. Retrieved November 11, 2020, from Railway-News: <https://railway-news.com/bt-cables-on-track-for-significant-business-growth/>
- Rajkumar. (2018, July 30). *Equivalence Partitioning Test Case Design Technique*. Retrieved December 22, 2020, from Software Testing Material: <https://www.softwaretestingmaterial.com/equivalence-partitioning-testing-technique/>
- Rapita Systems. (n.d.). *What is MC/DC?* Retrieved November 20, 2020, from Rapita Systems: <https://www.rapitasystems.com/mcdc-coverage>
- Red Hat. (n.d.). *What is CI/CD?* Retrieved December 17, 2020, from Red Hat: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
- Rehkopf, M. (n.d.). *What is Continuous Integration?* Retrieved December 17, 2020, from Atlassian: <https://www.atlassian.com/continuous-delivery/continuous-integration>
- Ricardo. (2015, July 17). *EN50126/8/9 Revisions*. Retrieved October 12, 2020, from Ricardo: <https://rail.ricardo.com/news/en50126-8-9-revisions>
- Rouse, M. (n.d.). *Static Analysis (static code analysis)*. Retrieved October 16, 2020, from TechTarget: <https://searchsoftwarequality.techtarget.com/definition/static-analysis-static-code-analysis>
- Sá, R. (2020, June 29). *CoT Guide*. Coimbra, Coimbra, Portugal: Critical Software.
- Sagal, V. (2020, November 30). *DO-178C: AFuzion Tells Us Why It's the Bible of Avionics Software Development*. Retrieved January 5, 2021, from NEWSTRAIL: <https://www.newstrail.com/do-178c-afuzion-tells-us-why-its-the-bible-of-avionics-software-development/>
- Saurabh. (2020, September 14). *What is Jenkins? | Jenkins For Continuous Integration | Edureka*. Retrieved from Edureka: <https://www.edureka.co/blog/what-is-jenkins/>
- Shah, H. (n.d.). *What is Functional Testing? Explained with Test Cases and Example (Updated)*. (SIMFORM) Retrieved October 14, 2020, from SIMFORM: <https://www.simform.com/functional-testing/>
- Sharma, L. (2019, November 21). *Boundary Value Analysis – A Black Box Testing Technique*. Retrieved October 26, 2020, from TOOLSQLA: <https://www.toolsqa.com/software-testing/istqb/boundary-value-analysis/>

- Siemens. (n.d.). *Siemens presents the world's first electric railway*. Retrieved October 16, 2020, from Siemens: <https://new.siemens.com/global/en/company/about/history/news/on-track.html>
- Silva, R. E. (2011). *Qualificação de Software Crítico*. Instituto Superior de Engenharia de Coimbra, Departamento de Engenharia Informática e de Sistemas. Coimbra: Instituto Politécnico de Coimbra. Retrieved October 13, 2020, from https://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/Teses/Tese_Mest_Ricardo-Silva.pdf
- Software, C. (2020, May 19). Critical Software. *Fabrico Internacional*. (J. Gabriel, Interviewer) RTP1. RTP, Lisboa. Retrieved October 12, 2020, from <https://www.rtp.pt/play/p6943/e473493/fabrico-internacional>
- Sousa, I. P. (2010). *ESTRATÉGIA EMPRESARIAL DA CRITICAL SOFTWARE*. ISCTE Business School, Departamento de Gestão. Lisboa: ISCTE Business School. Retrieved October 9, 2020, from <https://repositorio.iscte-iul.pt/bitstream/10071/3272/1/Tese%20Final%C3%ADssima.pdf>
- T, N. (2020, March 27). *Black-Box Testing*. Retrieved October 15, 2020, from Binary Terms: <https://binaryterms.com/black-box-testing.html>
- Techbaz. (n.d.). *C# Installation*. Retrieved February 9, 2021, from Techbaz: https://www.techbaz.org/Course/Cs_installation.php
- Terminiello, B. (n.d.). *Grécia Antiga*. Retrieved October 16, 2020, from Pinterest: <https://www.pinterest.fr/pin/443463894531186058/>
- Tew, S. (2015, August 11). *What is TCMS?* Retrieved December 17, 2020, from RailEngineer: <https://www.railengineer.co.uk/what-is-tcms/>
- The Railway Technical Website. (n.d.). *Rolling Stock Manufacture*. Retrieved November 4, 2020, from The Railway Technical Website: <http://www.railway-technical.com/trains/rolling-stock-manufacture.html>
- The Railway Technical_A. (n.d.). *Automatic Train Control*. Retrieved November 18, 2020, from The Railway Technical Website: <http://www.railway-technical.com/signalling/automatic-train-control.html>
- The Railway Technical_B. (n.d.). *Bogies*. Retrieved November 4, 2020, from The Railway Technical: <http://www.railway-technical.com/trains/rolling-stock-index-l/bogies.html>
- The Railway Technical_C. (n.d.). *Electric Traction Control*. Retrieved November 4, 2020, from The Railway Technical: <http://www.railway-technical.com/trains/rolling-stock-index-l/train-equipment/electric-traction-control-d.html>
- The Railway Technical_D. (n.d.). *Electro-Pneumatic Brakes*. Retrieved November 4, 2020, from The Railway Technical: <http://www.railway-technical.com/trains/rolling-stock-index-l/train-equipment/brakes/electro-pneumatic-brakes-d.html>
- The Railway Technical_E. (n.d.). *Train Equipment*. Retrieved November 4, 2020, from The Railway Technical: <http://www.railway-technical.com/trains/rolling-stock-index-l/train-equipment/>
- Things Solver. (n.d.). *Hello Docker*. Retrieved October 29, 2020, from Things Solver: <https://thingsolver.com/hello-docker/>
- Toshiba Railway India. (n.d.). *Toshiba Railway India*. Retrieved November 4, 2020, from Toshiba Railway India: <https://www.toshiba-india.com/railways.aspx>
- Train History. (n.d.). *History of Rail Transport*. Retrieved October 16, 2020, from Train History: <http://www.trainhistory.net/railway-history/railroad-history/>
- Turck, F. (2020, June 16). *The V-Model in Software Testing*. Retrieved March 7, 2021, from froglogic: <https://www.froglogic.com/blog/tip-of-the-week/the-v-model-in-software-testing/>
- TÜV SÜD. (n.d.). *Automotive Functional Safety according to ISO 26262*. Retrieved October 14, 2020, from TÜV SÜD: <https://www.tuvsud.com/en/industries/mobility-and-automotive/automotive-and-oem/iso-26262-functional-safety>
- Wikimedia Commons. (2020, June 14). *File:Critical Software Main Logo.jpg*. Retrieved October 9, 2020, from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Critical_Software_Main_Logo.jpg
- Wikipedia. (2018, February 20). *Salamanca (locomotive)*. Retrieved October 16, 2020, from Wikipedia: [https://en.wikipedia.org/wiki/Salamanca_\(locomotive\)](https://en.wikipedia.org/wiki/Salamanca_(locomotive))
- Wikipedia_A. (2020, September 2). *Critical Software*. Retrieved October 9, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Critical_Software
- Wikipedia_B. (2020, October 8). *Steam engine*. Retrieved October 16, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Steam_engine
- Wikipedia_C. (2020, April 17). *London Underground electric locomotives*. Retrieved October 16, 2020, from Wikipedia: https://en.wikipedia.org/wiki/London_Underground_electric_locomotives

- Wikipedia_D. (2020, September 17). *Diesel locomotive*. Retrieved October 16, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Diesel_locomotive
- Wikipedia_E. (2020, July 19). *Janney coupler*. Retrieved November 11, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Janney_coupler
- Wikipedia_F. (2021, January 29). *Apache Groovy*. Retrieved February 9, 2021, from Wikipedia: https://en.wikipedia.org/wiki/Apache_Groovy
- Wikipedia_G. (2021, March 30). *White-box testing*. Retrieved May 1, 2021, from Wikipedia: https://en.wikipedia.org/wiki/White-box_testing
- Wikipedia_H. (2020, October 4). *DO-178C*. Retrieved October 14, 2020, from Wikipedia: <https://en.wikipedia.org/wiki/DO-178C>
- Wikipedia_I. (2020, November 10). *Git*. Retrieved November 17, 2020, from Wikipedia: <https://en.wikipedia.org/wiki/Git>
- Wikipedia_J. (2020, September 8). *Functional testing*. Retrieved October 14, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Functional_testing
- Wikipedia_K. (2021, February 8). *Python (programming language)*. Retrieved February 9, 2021, from Wikipedia: [https://en.wikipedia.org/wiki/Python_\(programming_language\)#cite_note-AutoNT-7-29](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-7-29)
- Wollny, S. (2017, July 11). *Reliability, Availability, Maintainability, Safety (RAMS) and Life Cycle Costs (LCC)*. Retrieved October 12, 2020, from OTP: http://www.otp.go.th/uploads/tiny_uploads/Public/2560/04-April/10-25590915-EuropeRailStandard.pdf
- Wongm's Rail Gallery. (2017, June 5). *Empty carriage set at Southern Cross, doors on both sides of the train open*. Retrieved November 11, 2020, from Wongm's Rail Gallery: https://railgallery.wongm.com/vline-bits/F119_9161.jpg.html
- XpertLync. (n.d.). *ISO/IEC 27001 – Consultations*. Retrieved October 14, 2020, from XpertLync: <https://www.xpertlync.com/iso-iec-27001-consulting/>

Appendix

- **Appendix A**
 - **Basic Training - Processes & Tools Certificate**



Basic Training - Processes & Tools
Certificate number 154

Afonso Batista

successfully completed the Basic Training - Processes & Tools, thereby qualifying in the following topic areas:

- Project Management and Life Cycle
- Development methodologies
- Support activities
- Support tools
- Pull-request Exercise
- Innovation & Knowledge

Afonso Batista attended with an active participation in this program, achieving a Good Result in the final quiz.

Ama Catarina Fonseca.

23rd December 2020

○ **Basic Training - Software Development Process Certificate**



Basic Training - Software Development Process
Certificate number 170

Afonso Batista

successfully completed the Basic Training - Software Development Process, thereby qualifying in the following topic areas:

- SDP Overview
- EA introduction and CSW template
- Requirements Analysis – Concepts & Processes
- Requirements with EA
- Software Design - Concepts & Processes
- UML & UML with EA
- Software Construction - Concepts & Processes
- Software Testing - Concepts & Processes
- Hands-On Project with Scrum ceremonies

Afonso Batista attended with an active participation in this program, achieving a Good Result in the final quiz.

Ama Catarina Fonseca

23rd December 2020

• Appendix B

Jenkins file - complete file with some erasures.

```
node {
    properties([
        disableConcurrentBuilds(),
        parameters([
            string(name: 'BUILD_NUMBER', defaultValue: '||(NumBer)'),
        ])
    ])

    def ccuo, ccus, bb, directory
    def shouldDeleteWS = true

    //Array of Error codes
    def errArray=[]

    try{
        stage('Check branch name') {
            BRANCH_PATTERN = ~ //(Branch_Regex)

            def matcher = (BRANCH_NAME =~ BRANCH_PATTERN)
            if(matcher){
                //Debug data
                //confidential

                ccuo = matcher.group("ccuo")
                ccus = matcher.group("ccus")
                bb = ccuo ? ccuo : ccus

            }else{
                errArray << 1
                error "Branch does not follow the template"
            }
        }

        stage('Checkout repository') {
            echo '[INFO] Cloning repository'
            checkout([$class: 'GitSCM', branches: [[name: BRANCH_NAME]], doGenerateSubmoduleConfigurations: false, extensions: [[$class: 'CleanBeforeCheckout']], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '||(ID)', url: '||(GIT_REPOSITORY_URL)']]])
        }
    }
}
```

```

def buildName = "((//ProjectDATA))_${params.BUILD_NUMBER}"
currentBuild.description = "((//ProjectDATA))_${params.BUILD_NUMBE
R}"

stage('Pre-Build') {
    def buildsFolder = "./builds/${buildName}"
    def buildZip = ccuo != null ? "((//CCU-
O_BuildFile)" : "((//CCU-S_BuildFile)"

    if( fileExists("${buildsFolder}/${buildZip}") ){
        //Unzipping software build
        dir(buildsFolder){
            try {
                def unzipped = unzip zipFile: buildZip, quiet: tr
ue

                //Replace file
                if(ccuo) {
                    //((//Confidencial)
                } else {
                    //((//Confidencial)
                }
            } catch(Exception e){
                errArray << 2
                error "Failed to unzip test build"
            }
        }
        //((//Exception in Build Project)

        //Creating TestInfo for Test
        def testInfo
            //((//Confidencial)

        dir("./${directory}") {
            //((//Confidencial)
        }
    }else{
        errArray << 3
        error "Build ${params.BUILD_NUMBER} does not exists"
    }
}

```

```

stage('Build Projects') {
    dir("./${directory}") {
        try{
            powershell "${CSharpCompiler} Testcase.sln -
p:Configuration=Release"
        }catch(exception){
            errArray << 4
            error "Failed to compile ${directory}"
        }
    }
}
}catch(exception){
    //If build enters this block it already failed
    currentBuild.result = 'FAILED'
    //(///GIT_Notify)

    //Timeout throws an exception at end. This exception is caught i
n order to clean the WS instead of stoping the build
    try{
        timeout(time: 12, unit: 'HOURS') {
            if(!errArray.contains(1) || !errArray.contains(3)){
                def USER_INPUT = input(
                    message: 'Clean WORKSPACE - yes or no ?',
                    parameters: [
                        [$class: 'ChoiceParameterDefinition',
                            choices: ['no','yes'].join('\n'),
                            name: 'input',
                            description: 'WORKSPACE Clear']
                    ])
                if( "${USER_INPUT}" == "no"){
                    shouldDeleteWS = false
                    echo "WORKSPACE Not Clear"
                    error "Failed and workspace was not cleared"
                }
            }
        }
    }
}catch(Exception e) {
    echo "Timeout for user input ended. Default action of clean w
orkspace will be executed"
}
}

```

```
if(shouldDeletewS){  
    stage('Clean WORKSPACE'){  
        powershell 'Get-ChildItem . -Force | Remove-Item -Recurse -  
Force'  
        dir("${WORKSPACE}@script")  
        {  
            powershell 'Get-ChildItem . -Force | Remove-Item -  
Recurse -Force'  
        }  
    }  
}  
  
//If build was successful, GIT needs to be notified  
if(errArray.isEmpty()){  
    currentBuild.result = 'SUCCESS'  
    //(//GIT_Notify)  
}  
}
```


- **Appendix C**

This part of the attachment will be displayed only on the presentation, it contains confidential information from CSW and from his customer:

- **First TCD**

The requirement, Test Script, will be presented as a complement to the code and the TCD already presented in this dissertation.

- **Second TCD**

The requirement, Your Test Case Design & Test Script, and their results will be presented.

- **Third TCD**

The requirement, Your Test Case Design & Test Script, and their results will be presented.

- **Jenkins File**

Jenkins file presented in the context of the project will be, at this time, presented without erasures.

